On the Synthesis of Choreographies

Thesis submitted for the degree of

Doctor of Philosophy at the

University of Leicester

by

Julien Lange

Department of Computer Science

University of Leicester

August 2013

On the Synthesis of Choreographies

Julien Lange

Abstract. The theories based on session types stand out as effective methodologies to specify and verify properties of distributed systems. A key result in the area shows the suitability of choreography languages and session types as a basis for a choreography-driven methodology for distributed software development. The methodology it advocates is as follows: a team of programmers designs a global view of the interactions to be implemented (i.e., a choreography), then the choreography is projected onto each role. Finally, each program implementing one or more roles in the choreography is validated against its corresponding projection(s).

This is an ideal methodology but it may not always be possible to design one set of choreographies that will drive the overall development of a distributed system. Indeed, software needs maintenance, specifications may evolve (sometimes also during the development), and issues may arise during the implementation phase. Therefore, there is a need for an alternative approach whereby it is possible to infer a choreography from local behavioural specifications (i.e., session types).

We tackle the problem of synthesising choreographies from local behavioural specifications by introducing a type system which assigns – if possible – a choreography to set of session types. We demonstrate the importance of obtaining a choreography from local specifications through two applications. Firstly, we give three algorithms and a methodology to help software designers refine a choreography into a global assertion, i.e., a choreography decorated with logical formulae specifying senders' obligations and receivers' requirements. Secondly, we introduce a formal model for distributed systems where each participant advertises its requirements and obligations as behavioural contracts (in the form of session types), and where multiparty sessions are started when a set of contracts allows to synthesise a choreography.

Acknowledgements

First and foremost I would like to thank Emilio Tuosto for being such a great supervisor, friend and colleague over these past four years. Thank you very much Emilio it has been great working with you (and still is)!
I am grateful to my collaborators Laura Bocchi and Alceste Scalas. I am very glad I had the chance to work with both of you.
I would like to thank Mariangiola Dezani-Ciancaglini and Nir Piterman for examining my thesis, also for great comments and discussions during my viva.
I would like to thank my colleagues from Leicester who have had – in a way or another – some influence on my work: Daniela Petrişan whose "puzzle" became somehow part of this thesis; Tadeusz Litak with whom I had many discussions on early ideas on a type system from local to global types; Roy Crole and Fer-Jan de Vries for discussions and advice as part of my thesis committee.
I would like to thank the Department of Computer Science of the University of Leicester for offering me the chance of working and studying here.
I am grateful to Nobuko Yoshida and Pierre-Malo Deniélou for insightful discussions and comments.
Thank you to my other friends and (ex-)colleagues without whom working in the department would not be what it was. Thank you to my "brother-in-PhD" Kyriakos; and thank you to Gabriela, Octavian, Alex, Oliver, Sokkar, Muz, and all those that I am forgetting here, for being great friends in and outside the department.
I am grateful to my parents, sisters, and grand-parents for their support and encouragements during these past four years. Merci Angèle for visiting me whenever you could!
Last but not least, I would like to thank Franz Preud'homme who, fifteen years ago, gave me a book on programming with Delphi. Had he not been around, I probably would not have studied computer science.

To Angèle.

Contents

List of Figures

CHAPTER 1

Introduction

1.1 Motivations and Objectives

The design, implementation, and maintenance of distributed systems are not easy tasks. This is due, notably, to the fact that many parties – with different objectives and policies – may be involved in each one of these tasks. Also, the specification of such systems is generally done at different levels and possibly via different formalisms, e.g., each part of a system may be specified from internal and external perspectives, using different languages such as BPEL [61] and UML [62]. Addressing these tasks is made even more challenging when specifications change during the development cycle, when part of the system relies on legacy software, or when the system must evolve in a hostile environment, e.g., when a service relies on the dynamic discovery of other services, which might not be trusted.

In the last decade, message passing based distributed systems have been receiving much attention both from industry [64, 65, 67] and from the research community. From the research perspective, a major advantage is that such systems may be reasoned about via very well established theoretical tools such as the π-calculus [56]. The suitability of

these systems for theoretical work lead to a new approach to distributed software design and verification where types are not only used for checking, e.g., expressions and functions, but also to check that, e.g., the *behaviour* of a client matches the behaviour of a service offered by a server. Behavioural types, or *session types*, are used to assign an abstract behaviour to channels which are used by distributed programs to interact with one another.

Additionally, *choreography* languages such as WS-CDL [34] have been developed to model the interactions between different services from a global point of view. Notably, they allow software designers of different parts of a system to define – jointly – the rules of participation in the system, without delegating the control of their part of the system to another party. A groundbreaking work [44] relates choreography languages and session types to promote a choreography-driven methodology for distributed software development. The methodology it advocates, summarised in Figure 1.1, is as follows: a team of programmers designs a global view of the interactions to be implemented (i.e., a choreography), then the choreography is *projected* onto each role, or participant, finally each program implementing one or more roles in the choreography is *type-checked* against its corresponding projection(s). This approach permits, if the choreography satisfies some properties, to guarantee safety properties such as deadlock freedom and to ensure that an implementation abide by the specified protocol.

This is an ideal methodology, but it may not always be possible to design one set of choreographies that will drive the overall development of a distributed system. Indeed, software need maintenance, specifications may evolve (sometimes also during the development), and issues may arise during the implementation phase. Therefore, there is a need for an alternative approach whereby it is possible to infer a global point of view of a distributed system (i.e., a choreography) from local behavioural specifications (i.e., session types). Without such an approach the burden of (re-) constructing the choreographies of a distributed system is on the practitioners. Often, they may choose to omit this step,

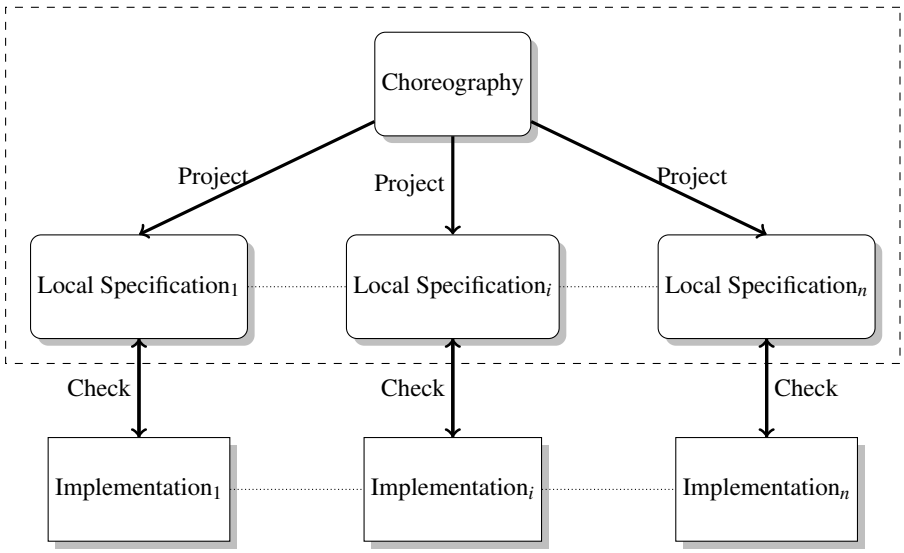


Figure 1.1: Choreography driven development

therefore creating a gap between an implementation and its specification. We propose to tackle the problem of synthesising choreographies from local behavioural specifications via a type system which assigns a choreography to a set of session types, if possible.

Obtaining a choreography from local specifications permits not only the methodology above to be applicable and thus to guarantee safety properties, but also allows to keep an implementation closer to its specification. Indeed, as the implementation progresses, one may infer choreographies from it and check that it still matches the specification. If there is a gap between implementation and specification, then it is possible to adapt both specification and implementation in a semi-automatic manner (using choreography projections and synthesis).

Keeping specifications and implementations close is not the only advantage of having a means to synthesise a choreography from local specifications. In fact, once a choreography has been obtained it may be refined into a more informative formalism, e.g, a *global assertion*, that is a choreography decorated with logical formulae specifying senders' obligations and receivers' requirements. We will see however that refining a choreography into a global assertion is not always a trivial task. We will give a few techniques to facilitate the task of the designer so that global assertions satisfy essential safety

properties.

In addition, synthesising choreographies may also be done at runtime to characterise a set of services, or contracts, which are fulfilling each other's requirements. We say that a set of contracts is compliant if it may be assigned a choreography. An advantage of having a choreography to represent the fact that a set of services is compliant is that it permits to reason about meta-level properties of the interactions, e.g., if the interactions always terminate or if the involved services will share some common properties once the interactions have terminated.

The main objectives of the thesis are as follows. Firstly, we tackle the problem of constructing a global view of a distributed system (i.e., a choreography) from local specifications: we focus on the top part of Figure 1.1 and reverse the "project" arrows. Secondly, we demonstrate the interest of obtaining a choreography from local specifications by (*i*) giving techniques to help refine a choreography into a global assertion and (*ii*) by introducing a model for distributed systems which use choreographies as a basis for combining services together.

1.2 Synopsis and Contributions

The thesis is divided in six chapters which we briefly summarise below.

- In the rest of this chapter, we list the published works on which the thesis is build.
- In Chapter 2, we introduce the context and foundations of our work and discuss related literature.
- In Chapter 3, we give a construction to synthesise a choreography from local specifications. This chapter represents the core contribution of this thesis: a type system which assigns a global type to a set of local types.

- In Chapter 4, we present three algorithms and a methodology to help designers decorate a choreography with logical predicate stating senders' obligations and receivers' requirements.
- In Chapter 5, we introduce a formal model for distributed systems where the synthesis of choreographies is used to define a compliance relation between contracts advertised by participants willing to cooperate.
- In Chapter 6, we conclude the thesis and discuss future research directions.

1.3 Publications

The thesis is based on the following publications (in chronological order). We describe their correspondance with the content of the thesis and the contributions of the author.

- **A Modular Toolkit for Distributed Interactions**. Julien Lange and Emilio Tuosto, *PLACES 2010*, [52].
  - *Author's contribution:* This is a paper of my own.
  - This work does not appear in the thesis, but the toolkit it describes was used (*i*) to identify the problem that Chapter 4 tackles (i.e., designing a *well-asserted* global assertion is not an easy task) and (*ii*) to test the algorithms presented in Chapter 4.
- **Amending Contracts for Choreographies**. Laura Bocchi, Julien Lange and Emilio Tuosto, *ICE 2011*, [13].
  - *Author's contribution:* In this work, my contributions include: part of the writing, part of the ideas and formalisations of the three algorithms; I had a major contribution in the proofs, the methodology, and the idea of the lifting algorithm.

  - *Corresponding part:* Chapter 4.

- **Three Algorithms and a Methodology for Amending Contracts for Choreographies**. Laura Bocchi, Julien Lange and Emilio Tuosto, *Sci. Ann. Comp. Sci., 22*, [14].

  - *Author's contribution:* This paper is an extented version of [13] above. This version includes all the proofs.

  - *Corresponding part:* Chapter 4 is essentially an extended and updated version of [14].

- **Synthesising Choreographies from Local Session Types**. Julien Lange and Emilio Tuosto, *CONCUR 2012*, [53].

  - *Author's contribution:* This is a paper of my own.

  - *Corresponding part:* Chapter 3 is major revision of [53]. The technicalities were updated in order to match recent developments on the relationship between communicating machines and session types. In addition, all the proofs are included in the chapter and many additional examples were added.

- **Choreography Synthesis as Contract Agreement**. Julien Lange and Alceste Scalas, *ICE 2013*, [51].

  - *Author's contribution:* In this work, my contributions include: part of the writing, part of the ideas and formalisations of our adaptation of CO_2, the variations of the fuse primitive, and the proofs.

  - *Corresponding part:* Chapter 5 is an extended version of [51], with all the proofs, additional examples and, notably, a larger discussion on variations of the fuse-primitive.

CHAPTER 2

Background

We review the key theories on which our work relies and discuss a few related approaches. We focus on (*i*) the basis on which our work is build – in Section 2.1 – and (*ii*) on works that study the relationship between global and local specifications – in Section 2.2.

2.1 Foundations

The theories based on session types stand out as effective methodologies to specify and verify properties of distributed systems. Amongst the research community, they have become a popular framework to reason about concurrent programs which communicate via (usually asynchronous) communication channels.

Since the first paper introducing session types as a means to design and verify properties of distributed systems in the mid-nineties, many extensions and applications have been proposed by many authors. These range from theoretical works establishing a Curry-Howard-like isomorphism between linear logic and session types [19, 20, 70], to works embedding session type theory into programming languages [45, 58–60], and industrial applications [64, 65].

In this section, we introduce the key elements of session type theory. We first present the theory of dyadic session types and its extension to multiparty session types. We then present two extensions which are essential for the next chapters: an extension which introduces design-by-contract for distributed interactions and an extension of the multiparty session types framework to communicating machines.

2.1.1 Dyadic Session Types

Session types were first introduced in [43, 68] by Honda et al. as a means to structure communications in concurrent programming. Analogously to data types in traditional programming language, where, e.g., type checking at compilation time ensures that the arguments of a function match its definition, dyadic session types guarantee that, e.g., the behaviour of a client-side application matches the behaviour of a service invoked on the server-side.

The theory relies on a few ingredients that we describe below. The key ingredient is the notion of *session*, that is a set of *structured* interactions which corresponds, for instance, to a communication protocol (from the communication initialisation to the termination). We give an example of a session in Example 2.3 below.

The framework introduced in [43] – and most of its extensions – is based on three main components: (*i*) a language used to write the processes which take part in the interactions (usually a variation of the π-calculus [56]); (*ii*) a syntax for *types* that describe the structure of the interactions (i.e., session types); and (*iii*) a type discipline which verifies that processes match their types and guarantees safety properties of the interactions. We describe informally each ingredient and illustrate how the theory works.

A session-oriented calculus. The first component is a calculus to model processes which interact via communication channels. In order to structure the interactions between parties, a few *interaction primitives* are made available to the programmers. They include

communication mechanisms à la π-calculus that deal with session channels as first-class values. We focus on the primitives and constructs of the language which directly relate to the rest of this thesis.

In the rest of this chapter, we let R range over process, n over public names, k over channels, and $\mathtt{e}$ over values.

$$
\begin{array}{rcll}
R & ::= & \overline{n}(k)\ \mathtt{in}\ R & \text{[SESSION REQUEST]} \\
& | & n(k)\ \mathtt{in}\ R & \text{[SESSION ACCEPTANCE]} \\
& | & k!\langle \mathtt{e} \rangle ; R & \text{[SEND]} \\
& | & k?(\mathtt{x}) ; R & \text{[RECEIVE]} \\
& | & k \lhd \mathtt{l} ;\ R & \text{[SELECT]} \\
& | & k \rhd \{\mathtt{l}_1 : R_1, \ldots, \mathtt{l}_n : R_n\} & \text{[BRANCH]} \\
& | & (\nu k) R & \text{[RESTRICTION]} \\
& | & R \mid R' & \text{[PARALLEL]} \\
& | & \ldots &
\end{array}
$$

Primitives [SESSION REQUEST] and [SESSION ACCEPTANCE] deal with session creation. The former indicates a request for a new session on public name n (i.e., a name known to other processes). Primitive [SESSION ACCEPTANCE] is its dual: it indicates the acceptance of a new session. These two primitives bind k in their respective continuations.

Example 2.1. A process implementing an FTP server would start with a primitive $n(k)$ $\mathtt{in}\ R$, indicating that it is waiting for a client request. Here n may be understood as *port 21*, k may be understood as the port number through which the data exchange will occur, and R is the implementation of the body of the server.

Similarly, a process implementing an FTP client would start with a primitive $\overline{n}(k)\ \mathtt{in}\ R$, indicating that the client sends a request to a server via port 21 (or n).[1] ◇

[1] Note that n may also be understood as a pair of IP address and port number.

Primitives [SEND] and [RECEIVE] allow processes to exchange data. The former indicates the sending of a value e on a channel k; while the latter indicates the reception of a datum from channel k, and binds x in the continuation R; once a value is received, x is replaced by that value in R.

Primitives [SELECT] and [BRANCH] provide a *select* and *branch* mechanism which allows participants to make an internal choice or to let the environment make a decision (external choice). Notably, the mechanism allows to provide a simple model of (distributed) method invocations, where a label l models the name of a method. Primitive [SELECT] indicates the sending of a label l on channel k; and primitive [BRANCH] indicates a choice made by another process, if label l_i is selected, then R_i is executed.

The construct [RESTRICTION] allows to restrict the use of channel k to the process R. Note that [RESTRICTION] is a runtime construct, i.e., it is not available to the programmer and appears only when sessions have been started. Concurrent processes are specified via construct [PARALLEL] which indicates the concurrent execution of processes R and R'.

In addition to these primitives, the language features standard constructions such as if-then-else, process definitions and invocations, terminated process $\mathbf{0}$ (trailing occurrences are often omitted), and special primitives for sending and receiving session names. These special primitives allow *delegation*, namely the fact that session channels may be communicated so to allow a process to delegate its "role" to another process.

We illustrate the syntax of the language in Example 2.2 below.

Example 2.2. Consider the two processes R_s and R_c below.

$$\begin{aligned} R_s \;=\; & n(k)\,\mathtt{in} \\ & k \rhd \{\;\; \mathsf{i} : k?(\mathsf{x})\,;\, k!\langle \mathsf{x} * 2\rangle, \\ & \qquad \mathsf{b} : k?(\mathsf{y})\,;\, k!\langle \neg\, \mathsf{y}\rangle \;\} \end{aligned}$$

The process R_s may be understood as a server which waits for a request on a public name

n, i.e., a name known by other processes in the system. Once a request has been received, the server offers two operations, or branches. If the branch i is selected, then the server receives an integer, and returns the double of this integer. If the branch b is selected, the server receives a boolean and returns its negation.

$$\begin{array}{rcl} R_c & = & \overline{n}(k)\ \mathtt{in} \\ & & \mathsf{if}(\ldots)\ \ \mathsf{then}\ k \lhd \mathsf{i}\,;\ k!\langle 3\rangle\,;\ k?(\mathsf{x}) \\ & & \qquad\qquad \mathsf{else}\ k \lhd \mathsf{b}\,;\ k!\langle \mathsf{true}\rangle\,;\ k?(\mathsf{y}) \end{array}$$

The process R_c is a client which, essentially, exhibits the dual behaviour. Once its session request has been accepted, it chooses either branch offered by the server – we abstract from the actual decision here. If the client chooses the i (resp. b) branch, it sends the value 3 (resp. true) via channel k, and expects an answer which will be stored in variable x (resp. y). ◇

The semantics of the calculus is similar to the one of the π-calculus (but for the session creation primitives) and we illustrate it with Example 2.3.

Example 2.3. Composing the two processes of Example 2.2 in parallel, we obtain $R_s \mid R_c$, which reduces to

$$(\nu k')\left(\begin{array}{l|l} k' \rhd \{\ \ \mathsf{i} : k'?(\mathsf{x})\,;\ k'!\langle \mathsf{x}*2\rangle, & \mathsf{if}(\ldots)\ \ \mathsf{then}\ k' \lhd \mathsf{i}\,;\ k'!\langle 3\rangle\,;\ k'?(\mathsf{x}) \\ \qquad\quad \mathsf{b} : k'?(\mathsf{y})\,;\ k'!\langle \neg \mathsf{y}\rangle\ \} & \qquad\quad \mathsf{else}\ k' \lhd \mathsf{b}\,;\ k'!\langle \mathsf{true}\rangle\,;\ k'?(\mathsf{y}) \end{array}\right)$$

where k' is a (fresh) channel restricted to these processes (i.e., no other process may use it).

At this point we can say that a *session* has started: two processes have started a new session by "meeting" via a public name n and they may now communicate "privately" via the fresh channel k'.

If the client chooses the i branch, the system reduces to

$$(\nu k')\left(k' \rhd \{\, \mathsf{i} : k'?(\mathsf{x})\,;\, k'!\langle \mathsf{x} * 2\rangle\,,\, \mathsf{b} : k'?(\mathsf{y})\,;\, k'!\langle \neg \mathsf{y}\rangle\,\} \quad | \quad k' \lhd \mathsf{i}\,;\, k'!\langle 3\rangle\,;\, k'?(\mathsf{x})\right)$$

and after a few steps, we obtain:

$$(\nu k')\left(k'!\langle 3 * 2\rangle \quad | \quad k'?(\mathsf{x})\right)$$

and the process terminates once the client has received the value 6 via channel k'. ◇

Session types. The next components of the theory are *session types*, also called local types, or end-point types. Essentially, a session type represents the abstract *behaviour* of a process on a given communication channel. We give their syntax below – adapted from [43] – and let P range over session types and a over message sorts.

$$\begin{array}{rcll} P & ::= & !\mathsf{a}\,;P & \text{[SEND]} \\ & | & ?\mathsf{a}\,;P & \text{[RECEIVE]} \\ & | & \oplus\{\mathsf{l}_1 : P_1, \ldots, \mathsf{l}_n : P_n\} & \text{[INTERNAL CHOICE]} \\ & | & +\{\mathsf{l}_1 : P_1, \ldots, \mathsf{l}_n : P_n\} & \text{[EXTERNAL CHOICE]} \\ & | & \mu\chi.P & \text{[RECURSION]} \\ & | & \chi & \text{[RECURSIVE CALL]} \\ & | & \mathbf{0} & \text{[END]} \end{array}$$

where a ranges over basic data types (or *sorts*) such as $\mathsf{bool}, \mathsf{int}$, etc.[2] There is a close correspondence with the primitives of the calculus presented above. Types [SEND] and [RECEIVE] correspond to the send and receive primitives. Types [INTERNAL CHOICE] and [EXTERNAL CHOICE] correspond the select and branch primitives. Note that type [INTERNAL CHOICE] says that

[2] These may also include session types, but we abstract from such constructions in this thesis.

any of the labels l_i may be *sent*, then behaviour P_i takes places. Types [RECURSION] and [RECURSIVE CALL] are used to represent recursive behaviours. Finally, type [END] indicates the termination of the session.

An important notion regarding dyadic session types is the one of *co-type*. The co-type of a type P, written $\overline{P}$, is defined as follows:

$$\begin{array}{lcllcllcl}
\overline{!\mathsf{a};P} & = & ?\mathsf{a};\overline{P} & \overline{\oplus\{\mathsf{l}_1:P_1,\ldots,\mathsf{l}_n:P_n\}} & = & +\{\mathsf{l}_1:\overline{P_1},\ldots,\mathsf{l}_n:\overline{P_n}\} & \overline{\mathbf{0}} & = & \mathbf{0} \\
\overline{?\mathsf{a};P} & = & !\mathsf{a};\overline{P} & \overline{+\{\mathsf{l}_1:P_1,\ldots,\mathsf{l}_n:P_n\}} & = & \oplus\{\mathsf{l}_1:\overline{P_1},\ldots,\mathsf{l}_n:\overline{P_n}\} & & & \\
\overline{\mu\chi.P} & = & \mu\chi.\overline{P} & \overline{\chi} & = & \chi & & &
\end{array}$$

Typing discipline. The final component of the framework is a *typing discipline* which guarantees safety properties of the system under consideration. One of the main contributions in [43] is a type system that, given a process R, permits to assign session types to the channels used by R. The reader does not need to have a precise understanding of the type system in the scope of this thesis, thus we only highlight the basic ideas.

The main judgement of the type system is of the form:

$$\Theta;\Gamma \vdash R \rhd \Delta$$

where Θ is an environment mapping process variables to processes (to deal with process definitions), environment Γ maps free names (e.g., public name n in Example 2.2) to pairs of local types, and expressions and variables to sorts. Environment Δ maps free channels (such as channel k in Example 2.2) to local types. We give a couple of typing rules to show the flavour of the type system.

$$\frac{\Theta;\Gamma \vdash R \rhd \Delta\cdot k:P}{\Theta;\Gamma\cdot n:\langle P,\overline{P}\rangle \vdash n(k)\ \mathtt{in}\ R \rhd \Delta} \qquad \frac{\Gamma \vdash \mathsf{e}:\mathsf{a} \qquad \Theta;\Gamma \vdash R \rhd \Delta\cdot k:P}{\Theta;\Gamma \vdash k!\langle \mathsf{e}\rangle;R \rhd \Delta\cdot k:!\mathsf{a};P}$$

The left-hand side rule validates session acceptance primitive; if the environment Γ maps a name n to a pair of dual local types and if the continuation is typable, with $k : P$ in environment Δ. Note that the rest of the derivation must ensure that the process uses the channel k as advertised by type P. The right-hand side rule validates sending actions; if the continuation is typable and the expression e is of type a.

Example 2.4. Continuing with Example 2.2, the type of channel k, from the point of view of R_s (resp. R_c) is given by P_s (resp. P_c) below.

$$P_s = +\{\mathsf{i} : ?\mathsf{int};!\mathsf{int}\ ,\ \mathsf{b} : ?\mathsf{bool};!\mathsf{bool}\} \qquad P_c = \oplus\{\mathsf{i} : !\mathsf{int};?\mathsf{int}\ ,\ \mathsf{b} : !\mathsf{bool};?\mathsf{bool}\}$$

It is easy to see that we have $\overline{P_c} = P_s$ and the public name n has type $\langle P_s, P_c \rangle$. $\diamond$

The main results of [43] are that (*i*) if a process is typable then it will never reduce to an error (e.g., a process receives string but was expecting a bool) and (*ii*) typing is preserved by reduction (*subject reduction*).

2.1.2 Multiparty Session Types

The initial theory of session types allowed only to reason about bi-party communications. In particular, the notion of *co-type* only makes sense for sessions that involve two participants. Following preliminary results in [24, 25], the dyadic framework was extended in [44] to a methodology to design, implement, and verify distributed software. The methodology – summarised in Figure 2.1 – advocates that (*i*) a team of programmers specify a global description of the protocol, i.e., a *choreography* (or global type). Then, (*ii*) the choreography is *projected* onto each *participant* (or role) so to obtain a *local type*, i.e., a description of the role that the participant plays in the session. Finally, (*iii*) processes are type-checked to ensure that they validates their respective local types.

In the rest of this section, we present the main ideas introduced in [44], using the

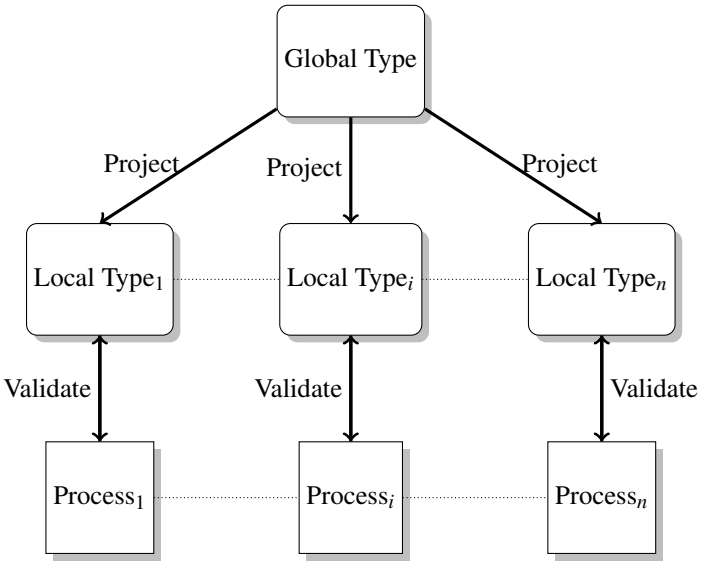


Figure 2.1: The multiparty session types approach

simplified syntax from [11].

Remark 2.1. *We will only briefly discuss the processes from which local types are extracted from. In this thesis, we focus on the relationship between global and local types, i.e., the top part of Figure 2.1. The extraction of session types from processes has been studied extensively in, e.g., [31, 43, 44].*

Choreographies. Choreographies are specified by global types. Intuitively, a global type establishes the interaction pattern for the harmonious coordination of distributed parties. The syntax of global types is given by the following productions:

$$
\begin{array}{rcll}
\mathcal{G} & ::= & \mathtt{s} \to \mathtt{r} : \mathtt{a}; \mathcal{G}' & \text{[VALUE PASSING]} \\
& | & \mathtt{s} \to \mathtt{r} : \{\mathtt{l_i} : \mathcal{G}_i\}_{i \in I} & \text{[BRANCHING]} \\
& | & \mathcal{G}' \mid \mathcal{G}'' & \text{[PARALLEL]} \\
& | & \mu\chi.\mathcal{G}' & \text{[RECURSION]} \\
& | & \chi & \text{[RECURSIVE CALL]} \\
& | & \mathbf{0} & \text{[END]}
\end{array}
$$

Type [VALUE PASSING] says that participant $\mathtt{s}$ sends a message (of sort $\mathtt{a}$) to participant $\mathtt{r}$, then the interactions in $\mathcal{G}'$ take place. Type [BRANCHING] says that participant $\mathtt{s}$ sends one of the labels l_i to participant $\mathtt{r}$. If l_j is sent then the interactions in $\mathcal{G}_j$ take place. Type [PARALLEL] represents the concurrent run of the interactions described in $\mathcal{G}'$ and $\mathcal{G}''$. Type [RECURSION] is a recursive type for recursive conversation structure, assuming that type variables ([RECURSIVE CALL]) are guarded in the standard way. Type [END] represents the termination of the session.

We illustrate the theory via Examples 2.5, 2.6, and 2.7 adapted from [44].

Example 2.5. Below is a global type specifying a protocol with two buyers ($\mathtt{b}_1$ and $\mathtt{b}_2$) and a seller ($\mathtt{s}$). The buyers $\mathtt{b}_1$ and $\mathtt{b}_2$ want to purchase a book from $\mathtt{s}$ by combining their funds.

$$
\begin{aligned}
\mathcal{G}_{\text{ex2.5}} \;=\;& \mathtt{b}_1 \to \mathtt{s} : \mathsf{string}; && (1)\\
& \mathtt{s} \to \mathtt{b}_1 : \mathsf{int}; && (2)\\
& \mathtt{b}_1 \to \mathtt{b}_2 : \mathsf{int}; && (3)\\
& \mathtt{b}_2 \to \mathtt{s} : \{\ \mathsf{ok} : \quad \mathtt{b}_2 \to \mathtt{s} : \mathsf{string}; && (4)\\
& \qquad\qquad\qquad\quad \mathtt{s} \to \mathtt{b}_2 : \mathsf{date}; \mathbf{0}, && (5)\\
& \qquad\qquad \mathsf{quit} : \quad \mathbf{0}\ \} && (6)
\end{aligned}
$$

In (1), $\mathtt{b}_1$ and $\mathtt{s}$ interact and exchange a book title. In (2), $\mathtt{s}$ sends a quote to $\mathtt{b}_1$. In (3), $\mathtt{b}_1$ tells $\mathtt{b}_2$ its contribution to the purchase. In (4-6), $\mathtt{b}_2$ may refuse (selecting label quit) or accept the deal (selecting label ok). In the former case, the protocol terminates there, otherwise it continues as in (4-5), namely $\mathtt{b}_2$ and $\mathtt{s}$ exchange delivery address and date. ◇

Remark 2.2. *In [44], interactions specify a channel over which participants communicate. For simplicity, we use to syntax of [11] and omit channel identities. This approach is also reflected in recent works such as [33, 39, 40, 73].*

When interactions specify channels, global types are required to satisfy a property

called linearity *[44]. Linearity essentially ensures that no race may occur on a channel, i.e., each message put on a channel may be received by exactly one participant.*

Local types & projections. Local types are similar to those presented in the previous section, except that they now specify to (resp. from) whom a message is sent (resp. expected). The syntax of (multiparty) *local types* is as follows

$$
\begin{array}{rcll}
P & ::= & \mathtt{r}!\mathsf{a};P & \text{[SEND]} \\
 & | & \mathtt{s}?\mathsf{a};P & \text{[RECEIVE]} \\
 & | & \mathtt{r}\oplus\{\mathsf{l}_1 : P_1, \ldots, \mathsf{l}_n : P_n\} & \text{[INTERNAL CHOICE]} \\
 & | & \mathtt{s}+\{\mathsf{l}_1 : P_1, \ldots, \mathsf{l}_n : P_n\} & \text{[EXTERNAL CHOICE]} \\
 & | & \mu\chi.P & \text{[RECURSION]} \\
 & | & \chi & \text{[RECURSIVE CALL]} \\
 & | & \mathbf{0} & \text{[END]}
\end{array}
$$

For instance, the type [INTERNAL CHOICE] says that one of the labels l_i may be sent to $\mathtt{r}$, then the interactions P_i take place.

If it satisfies certain conditions – which we will discuss in length in Chapter 3 – a global type may be *projected* onto each participant of the choreography. The projection is a simple syntactic operation which, given a global type $\mathcal{G}$ and a participant $\mathtt{s}$, returns a local type; we write $\mathcal{G}\upharpoonright_{\mathtt{s}}$ for the projection of $\mathcal{G}$ onto participant $\mathtt{s}$.

Example 2.6. We give the local types obtained by projecting the global type $\mathcal{G}_{\text{ex2.5}}$ of Example 2.5:

$$
\begin{array}{rcl}
\mathcal{G}_{\text{ex2.5}}\upharpoonright_{\mathtt{b}_1} & = & \mathtt{s}!\mathsf{string};\mathtt{s}?\mathsf{int};\mathtt{b}_2!\mathsf{int};\mathbf{0} \\
\mathcal{G}_{\text{ex2.5}}\upharpoonright_{\mathtt{b}_2} & = & \mathtt{b}_1?\mathsf{int};\mathtt{s}\oplus\{\mathsf{ok} : \mathtt{s}!\mathsf{string};\mathtt{s}?\mathsf{date};\mathbf{0}\,,\ \mathsf{quit} : \mathbf{0}\} \\
\mathcal{G}_{\text{ex2.5}}\upharpoonright_{\mathtt{s}} & = & \mathtt{b}_1?\mathsf{string};\mathtt{b}_1!\mathsf{int};\mathtt{b}_2+\{\mathsf{ok} : \mathtt{b}_2?\mathsf{string};\mathtt{b}_2!\mathsf{date};\mathbf{0}\,,\ \mathsf{quit} : \mathbf{0}\}
\end{array}
$$

The meaning of each type is straightforward, knowing the global type of Example 2.5. For instance, the type $\mathcal{G}_{\text{ex2.5}}\upharpoonright_{\mathtt{s}}$ says that the participant waits for a message of type string from participant $\mathtt{b}_1$ and replies with a message of type int. Then it let $\mathtt{b}_2$ choose either the quit branch, in which case its behaviour is finished; or the ok branch, in which case it receives a string from $\mathtt{b}_2$ and sends back a date. $\diamond$

Session calculus. The main differences between the syntax of *processes* in the dyadic setting and the one in the multiparty setting are as follows. First, the semantics now adopts asynchronous communications, i.e., the sending of messages is a non-blocking operation. Second, the primitives for session request and acceptance now specify the role a process is playing in the session. We illustrate the syntax and semantics of the calculus in Example 2.7. below.

Example 2.7. Consider the three processes below, implementing the protocol specified by the global type $\mathcal{G}_{\text{ex2.5}}$ in Example 2.5.

$$
\begin{aligned}
R_{\mathtt{b}_1} &= \overline{n}_{[\mathtt{b}_1]}(k)\ \mathtt{in}\ k!\mathtt{s}\langle\text{"Loving Sabotage"}\rangle;\, k?\mathtt{s}(\mathsf{x});\, k!\mathtt{b}_2\langle \mathsf{x}/2\rangle \\
R_{\mathtt{b}_2} &= n_{[\mathtt{b}_2]}(k)\ \mathtt{in}\ k?\mathtt{b}_1(\mathsf{y});\, \mathsf{if}(\mathsf{y} > 50)\ \mathsf{then}\ k \lhd \mathtt{s} : \mathsf{quit};\, \mathbf{0} \\
&\qquad\qquad \mathsf{else}\ k \lhd \mathtt{s} : \mathsf{ok};\, k!\mathtt{s}\langle\text{"Leicester"}\rangle;\, k?\mathtt{s}(\mathsf{y}') \\
R_{\mathtt{s}} &= n_{[\mathtt{s}]}(k)\ \mathtt{in}\ k?\mathtt{b}_1(\mathsf{z});\, k!\mathtt{b}_1\langle 70\rangle;\, k \rhd \mathtt{b}_2\{\, \mathsf{quit} : \mathbf{0}, \\
&\qquad\qquad \mathsf{ok} : k?\mathtt{s}(\mathsf{z}');\, k!\mathtt{s}\langle \mathsf{today}\rangle\,\}
\end{aligned}
$$

The process $R_{\mathtt{b}_1}$ is the implementation of the first buyer and is the one requesting the creation of a new session. Once the session has been established, $\mathtt{b}_1$ sends the title of a book to the seller $\mathtt{s}$, along channel k, $\mathtt{b}_1$ then expects a quote from the seller. Finally, $\mathtt{b}_1$ sends the amount of money it is willing to share with $\mathtt{b}_2$. The process $R_{\mathtt{b}_2}$ is the implementation of the second buyer. When $\mathtt{b}_2$ has accepted the session request, $\mathtt{b}_2$ receives the price that $\mathtt{b}_1$ is willing to pay then decides whether or not to proceed. If the price is greater than

50, $\mathtt{b}_2$ aborts the purchase, otherwise, the buyer sends its address and expects and answer from the seller. The process $R_{\mathtt{s}}$ is the implementation of the seller and is essentially symmetric to the other two processes. Here, we abstract from the way $\mathtt{s}$ retrieves the price associated with a book and assume that all books cost 70, for simplicity.

The composition of these processes reduces as

$$R_{\mathtt{b}_1} \mid R_{\mathtt{b}_2} \mid R_{\mathtt{s}} \quad \rightarrow \quad (\nu k')\ \left(R'_{\mathtt{b}_1} \mid R'_{\mathtt{b}_2} \mid R'_{\mathtt{s}}\right)$$

where k' is a (fresh) channel private to these three processes and $R'_{\mathtt{b}_1}$, $R'_{\mathtt{b}_2}$, and $R'_{\mathtt{s}}$ are as follows.

$$\begin{aligned}
R'_{\mathtt{b}_1} &= k'[\mathtt{b}_1]\,!\,\mathtt{s}\langle\text{"Loving Sabotage"}\rangle;\ k'[\mathtt{b}_1]\,?\,\mathtt{s}\,(\mathsf{x});\ k'[\mathtt{b}_1]\,!\,\mathtt{b}_2\langle \mathsf{x}/2\rangle \\
R'_{\mathtt{b}_2} &= k'[\mathtt{b}_2]\,?\,\mathtt{b}_1\,(\mathsf{y});\ \mathsf{if}(\mathsf{y} > 50)\ \mathsf{then}\ k'[\mathtt{b}_2] \lhd \mathtt{s} : \mathsf{quit};\ \mathbf{0} \\
&\qquad\qquad \mathsf{else}\ k'[\mathtt{b}_2] \lhd \mathtt{s} : \mathsf{ok};\ k'[\mathtt{b}_2]\,!\,\mathtt{s}\langle\text{"Leicester"}\rangle;\ k'[\mathtt{b}_2]\,?\,\mathtt{s}\,(\mathsf{y}') \\
R'_{\mathtt{s}} &= k'[\mathtt{s}]\,?\,\mathtt{b}_1\,(\mathsf{z});\ k'[\mathtt{s}]\,!\,\mathtt{b}_1\langle 70\rangle;\quad k'[\mathtt{s}] \rhd \mathtt{b}_2\{\ \mathsf{quit} : \mathbf{0},\ \\
&\qquad\qquad \mathsf{ok} : k'[\mathtt{s}]\,?\,\mathtt{s}\,(\mathsf{z}');\ k'[\mathtt{s}]\,!\,\mathtt{s}\langle\mathsf{today}\rangle\}
\end{aligned}$$

Observe that each instance of channel k has been replaced by a channel of the form, e.g., $k'[\mathtt{b}_2]$. For instance $k'[\mathtt{b}_2]\,?\,\mathtt{b}_1\,(\mathsf{y})$ says that participant $\mathtt{b}_2$ expects a message from $\mathtt{b}_1$ on the session channel k'.[3] Similarly, the prefix $k'[\mathtt{b}_2] \lhd \mathtt{s} : \mathsf{quit}$ says that participant $\mathtt{b}_2$ sends the label quit to participant $\mathtt{s}$, via channel k'. ◇

Typing & results. Typing processes in a multiparty setting is quite similar to the binary case. The main difference being that instead of checking whether a public name n is assigned a pair of dual types, one checks that a public name is typed by a global type and that each process taking part in the session may be typed by one of the projections of the

[3] The semantics allows for message from different sender to be re-ordered within a channel such as k'.

global type. For instance, the rule for typing session acceptance is of the form:

$$\frac{\Gamma \vdash n : \langle \mathcal{G} \rangle \qquad \Theta; \Gamma \vdash R \rhd \Delta \cdot k : \mathcal{G}\!\upharpoonright_{\mathtt{s}}}{\Theta; \Gamma \vdash n_{[\mathtt{s}]}(k) \ \mathtt{in}\ R \rhd \Delta}$$

which requires that n is of global type $\mathcal{G}$ and that the way participant $\mathtt{s}$ uses channel k must match the projection of $\mathcal{G}$ onto $\mathtt{s}$.

The main result in [44] is that, for typable processes, *progress* is guaranteed within a session, i.e., given a set of processes involved in a session (which is not interleaved with any other session), these processes will either be able to reduce further or terminate. Another result shows that the behaviour of a typable process will always follow the interactions specified by the global type, this property is called *session fidelity*.

2.1.3 Design-by-Contract for Distributed Multiparty Interactions

The multiparty sessions types framework was extended in [12], applying the ideas of Design-by-Contract [55] to the specification and verification of distributed systems.

The main new notion is the one of *global assertions* which decorate global types with logical formulae (*predicates*) that constrain interactions, declaring senders' obligations and receivers' requirements on the values of the exchanged data and on the choice of the branches to follow. This adds fine-grained constraints to the specification of the interaction structure.

Example 2.8. Consider for instance the *global assertion* below, where the values of the messages are represented by the *interaction variables* x and y.

$$\begin{aligned} \mathcal{G}_{\text{ex2.8}} \quad = \quad & \mathtt{Alice} \to \mathtt{Bob} : \{x \,|\, x > 0\}; \\ & \mathtt{Bob} \to \mathtt{Carol} : \{y \,|\, y > x\} \end{aligned}$$

$\mathcal{G}_{\text{ex2.8}}$ describes a protocol with three participants, `Alice`, `Bob`, and `Carol`, who agree on a "contract" constraining the interaction variables x and y. The contract stipulates that (*i*) `Alice` has to send `Bob` a positive value x in the first interaction, and that (*ii*) `Bob` must send `Carol` a value y strictly greater than x, fixed by `Alice` in the first interaction. Notice that `Bob` can fulfill his pledge (i.e., the predicate $y > x$ in the second interaction above) only after he has received the value x from `Alice`. ◇

Similarly to a global type, a global assertion $\mathcal{G}$ may be projected on *endpoint assertions* that are local types constrained according to the predicates in $\mathcal{G}$. For instance, the projection on `Alice` in $\mathcal{G}_{\text{ex2.8}}$ is an endpoint assertion prescribing that `Alice` has to send a positive value to `Bob`. Endpoint assertions can be used for static validation of the actual processes implementing one or more roles in a choreography represented by $\mathcal{G}$, and/or to synthesise monitor processes for runtime checking/enforcement [32].

The calculus of [44] is augmented with predicates for checking that the sent or received values validate the predicates so to obtain the following primitives:

$$k!\langle \mathsf{e} \rangle(x)\{\psi\};R \quad \text{[SEND]} \qquad\qquad k?(x)\{\psi\};R \quad \text{[RECEIVE]}$$

where e is a value, x a variable and ψ a predicate. The idea is that, in the [SEND] case, if $\psi\{\mathsf{e}/x\}$ evaluates to true, then the continuation R is executed, otherwise the process reduces to an error. Similarly, in the [RECEIVE] case, the value received by the process at runtime, say e, must be so that $\psi\{\mathsf{e}/x\}$ evaluates to true, otherwise the process reduces to an error.

If a global assertion is *well-asserted*, namely when it obeys two precise design principles: *history-sensitivity* and *temporal satisfiability*, it may be projected, and processes may be type-checked against the projections. If type-checking succeeds then the system is guaranteed to be error free.

Informally, history-sensitivity demands that a party having an obligation on a predi-

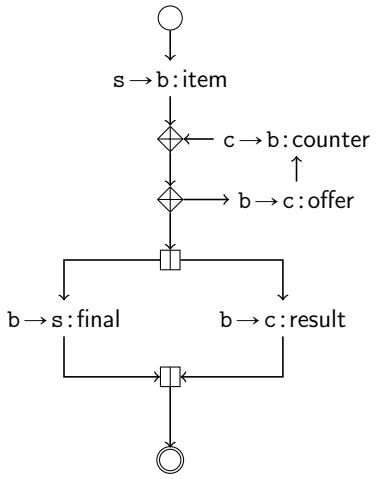


Figure 2.2: Example of generalised global type

cate has enough information for choosing a set of values that guarantees it. Instead, temporal satisfiability requires that the values sent in each interaction do not make predicates of future interactions unsatisfiable.

Remark 2.3. *We give a more precise account of global assertions, history-sensitivity, and temporal satisfiability in Chapter 4.*

2.1.4 Session Types and Communicating Machines

In [40], Deniélou and Yoshida introduced *generalised global types*, extending significantly the expressiveness of the global types of [44] (cf. Section 2.1.2) and characterising a relationship between communicating machines and multiparty session types. Below, we highlight the main novelties introduced in this work which we will refer to later in this thesis.

Overview. Generalised global types allow to describe protocols via general graphs, instead of a tree-like structure. Branching constructs, parallel operator and recursive definition are replaced by gates and edges which connect interactions of the form $\mathtt{s} \to \mathtt{r} : \mathtt{a}$. A $\oplus$-gate indicates either a choice branch, merge, or a recursion; a $\boxplus$-gate indicates either

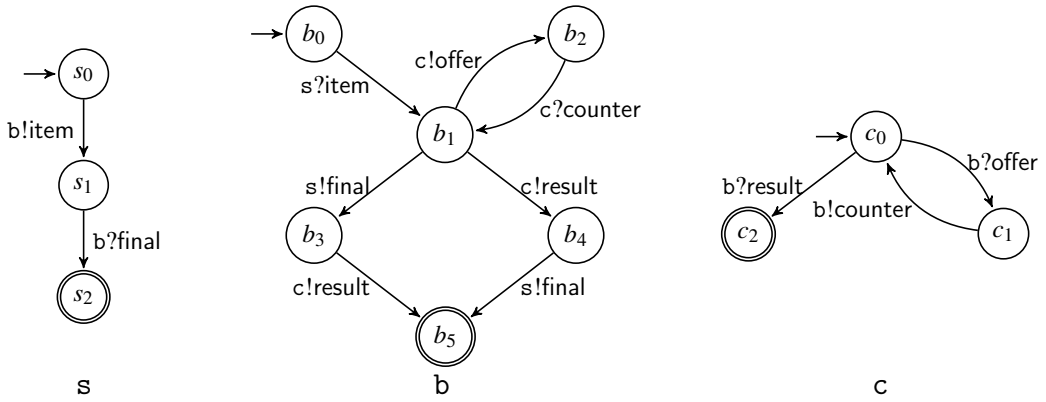


Figure 2.3: Example of communicating machines

fork or join of concurrent threads. We give an example of a generalised global type in Example 2.9 – borrowed from [40].

Example 2.9. The global type in Figure 2.2 specifies a protocol where "a seller (s) relies on a broker (b) to negotiate and sell an item to a client (c)" [40]. First, s sends a message item to b. The broker then chooses whether to (*i*) enter a loop where the broker and the client negotiate via offer and counter-offer messages; or (*ii*) the broker accepts the sale and terminates the protocol by sending concurrently messages final and result to the seller and the client, respectively. ◇

Interestingly, if a generalised global type satisfies some conditions it may be projected onto local types that are, essentially, *communicating machines* – or communicating finite states machines (CFSM) [16]. This permits to identify a new class of CFSMs, called Multiparty Session Automata.

A system of communicating machines is a list of automata which communicate via unbounded FIFO channels, generally assuming that, given two machines, they communicate via a pair of distinguished uni-directional channels. We give an example of a system of CFSMs in Example 2.10 – again borrowed from [40].

Example 2.10. The machines in Figure 2.3 correspond to the projections of the global

type of Example 2.9. A transition b!item in machine s says that s may put a message item on the queue from s to b. Once such a message is on the queue, machine b may fire its transition – s?item – thus removing the message from the queue. ◇

The authors also give a translation to/from traditional local types (such as the ones in Section 2.1.2) to communicating machines. Remarkably, these local types may now feature concurrent threads. For instance, the local type corresponding to machine b in Figure 2.3 explicitly shows that the actions s!final and c!result are executed concurrently.

Finally, a new session calculus is introduced in [40] so to allow these extensions to be fully supported at the programming language level.

Remark 2.4. *Global types, local types, and calculus in [40] are formally defined in terms of a set of mutually recursive variables which may be depicted as graphs (or automata) as shown in this section. Here, we only use the graphical notation for the sake of clarity.*

Safety properties. For a generalised global type to be projected onto "safe" local types, it must satisfy two essential conditions: *local choice* and *linearity*.

- The local choice condition requires the following for each choice in a global type.
  - There is exactly one participant who makes the decision, i.e., for each $\oplus$-gate with several outgoing edges, the first action of all the branches should have the same sender.
  - The choice must be propagated to the other participants involved in the choice, i.e., it cannot be the case that a participant behaves differently in two branches without knowing which branch was chosen by the sender.
- The linearity condition ensures that no races occur within the process of a participant. To achieve this, the authors "impose that no participant can be faced with two concurrent receptions where messages can have the same label" [40].

In addition, the authors require the generalised global types to be "sane", e.g., there should be exactly one starting point in the choreography, there should be at most one finishing point, and all threads must be collected by a join.

When these conditions are satisfied, the global type may be projected onto its participant. The system of CFSMs thus obtained is then guaranteed to satisfy the usual safety and liveness properties. In particular, the projections are deadlock free and all messages sent are eventually received. We discuss these properties in more details in Section 3.3.2.

2.2 Related Work

We discuss recent works that tackle the design and verification of distributed systems and focus on those that are based either on session type theory or on choreographies. In Section 2.2.1, we present different approaches that formalise the relationship between global and local specifications. In Section 2.2.2, we discuss works that tackle the problem of inferring a global specification from local ones. In Section 2.2.3, we report a few major extensions of the theories based on session types. In Section 2.2.4, we discuss briefly a few approaches that tackle the verification of (multiparty) session-based distributed systems without choreographies.

2.2.1 Global versus Local Specifications

A main concern in choreography-driven development is that the projections of a choreography should respect the structure of the interactions specified by the choreography. We distinguish three main lines of work that tackle this problem. First, Castagna et al. [28, 29] give a presentation of global types as a formal language and relate global and local types via a trace semantics. Second, Lanese et al. [17, 47, 50] give different notions of *conformance* that specify whether the projections of a choreography indeed match the behaviour specified by the choreography. Third, Bultan et al. [9, 10] focus on the notion of realis-

ability. Essentially, a choreography is realisable if "it is possible to build a distributed system that communicates exactly as the choreography specifies" [9].

Global types and multiparty sessions. In [28, 29], the authors introduce a variation of global types, give them a trace semantics, and discuss their properties with respect to their projections. We survey the part of this work that is of specific interest in the scope of this thesis.

The authors allow general sequencing in global types, e.g., types of the form $(\mathcal{G}_1 \mid \mathcal{G}_2)$; $(\mathcal{G}_3 \mid \mathcal{G}_4)$, but restrict recursive global types to iterations, i.e., global types of the form $\mathcal{G}^*$ – the usual Kleene star. This allows them to define global types as a traditional formal language and have a quite natural definition of *traces* of global types (and local types).

They identify a few basic properties that an implementation of a global type should guarantee, we describe them below.

Sequentiality says that if a global type specifies an order between interactions, then all executions of its implementation must respect this ordering. For instance, in the global type:

$$\mathtt{s}_1 \to \mathtt{r}_1 : \mathtt{a};\ \mathtt{s}_2 \to \mathtt{r}_2 : \mathtt{b}$$

if $\{\mathtt{s}_1, \mathtt{r}_1\} \cap \{\mathtt{s}_2, \mathtt{r}_2\} = \varnothing$ it is not possible for an implementation to guarantee that these two interactions will be executed in the prescribed order.

Alternativeness states that if a global type specifies an exclusive choice between two sets of interactions, then all the executions of its implementation must exhibit exactly one of these sets.

Shuffling states that all the executions of the implementation of a global type must exhibit some shuffling of the actions prescribed by the global type. For instance,

consider:

$$\mathtt{s_1} \rightarrow \mathtt{r_1}:\mathtt{a} \quad | \quad \mathtt{s_2} \rightarrow \mathtt{r_2}:\mathtt{b}$$

if an implementation of this global type exhibits the action on a but not the one on b, then it violates the specification.

Fitness says that all the sequences exhibited by an implementation of a global type must "fit" the interactions prescribed by the global type. Considering the global type

$$\mathtt{s} \rightarrow \mathtt{r}:\mathtt{a} \quad + \quad \mathtt{s} \rightarrow \mathtt{r}:\mathtt{b} \tag{2.1}$$

if an implementation exhibits an action unspecified by (2.1), say c, then the fitness condition is violated, since the action c is not prescribed by the global type.

Exhaustivity states that if a sequence of interactions is specified by a global type, then there must exist at least one execution of its implementation that exhibits these interactions (up-to possible re-ordering). For instance, an implementation of (2.1) such that s never sends b violates the condition.

Besides the global types violating the basic condition on sequentiality, the authors identify two kinds of "more serious" flaws, which may be illustrated as follows. The global type in (2.2) specifies a choreography where participant n has to behave differently in each branch of the choice, but is unaware of which branch was selected by s.

$$\mathtt{s} \rightarrow \mathtt{r}:\mathtt{a};\ \mathtt{r} \rightarrow \mathtt{n}:\mathtt{a};\ \mathtt{n} \rightarrow \mathtt{s}:\mathtt{a} \quad + \quad \mathtt{s} \rightarrow \mathtt{r}:\mathtt{b};\ \mathtt{r} \rightarrow \mathtt{n}:\mathtt{a};\ \mathtt{n} \rightarrow \mathtt{s}:\mathtt{b} \tag{2.2}$$

The global type in (2.3) specifies a choreography where the choice is "distributed" between participants s and n. This sort of choreography cannot be implemented without

covert channels to allow participants to agree on which branch to choose.

$$\mathtt{s} \to \mathtt{r} : \mathtt{a} \quad + \quad \mathtt{r} \to \mathtt{s} : \mathtt{b} \tag{2.3}$$

Remarkably, the authors are able to characterise these issues in terms of an ordering on the traces of global types and traces of local types.

Remark 2.5. *Observe that the* local choice *condition of [40] excludes problematic global types such as* (2.2) *and* (2.3).

Another important contributions of this work, is a novel (stronger) notion of progress which, essentially, states that, given an implementation of a global type, every participant must be able to (eventually) reach a terminated state. This is stronger that the progress of, e.g, [44] which only requires that the implementation (as a whole) is always able to reduce further or terminate.

Conformance. In [50], the authors identify two approaches to choreographies. The first approach, called *interaction-oriented*, is best represented by choreography languages such as WS-CDL [34] and the global types of Section 2.1.2. In our setting, we may consider interaction-oriented choreographies (IOC) as global types. For instance, adapting the first example of [50] to our syntax we have that

$$\mathtt{s}_1 \to \mathtt{r}_1 : \mathtt{a}; \mathtt{s}_2 \to \mathtt{r}_2 : \mathtt{b} \tag{2.4}$$

is a simple IOC describing that roles $\mathtt{s}_1$ and $\mathtt{r}_1$ interact on an operation $\mathtt{a}$, then roles $\mathtt{s}_2$ and $\mathtt{r}_2$ interact on an operation $\mathtt{b}$.

The second approach, called *process-oriented*, is best illustrated by BPEL4Chor [38] (extending WS-BPEL [61] to choreographies). In our setting, process-oriented chore-

ographies (POC) may be represented by systems of local types; for example

$$\mathtt{s}_1[!\mathtt{a}] \mid \mathtt{r}_2[?\mathtt{a}] \mid \mathtt{s}_2[!\mathtt{b}] \mid \mathtt{r}_2[?\mathtt{b}] \tag{2.5}$$

corresponds to the IOC of (2.4), i.e., system (2.5) may be seen as the projections of the choreography in (2.4). In fact, these two "choreographies" may exhibit different behaviours, e.g., there is no ordering between the actions on a and those on b in (2.5).

The authors define two languages, one for IOC and one for POC. Both languages consist of essentially the same constructs: sequential and parallel compositions, non-deterministic choice, empty process, and termination. The basic construct in the IOC language is the interaction, i.e., the construct $\mathtt{s} \to \mathtt{r} : \mathtt{a}$ corresponds to an interaction between s and r on an operation a. The basic constructs in the POC language are output !a and input ?a. The semantics of the POC language is given in two versions: synchronous and asynchronous.

The relationship between an IOC and a POC differs depending on the communication models one considers. The authors identify four possible semantics that we describe below.

- The *synchronous semantics* demands that the POC behaves as specified by the IOC, following a synchronous semantics (i.e., corresponding send and receive occur at the same time). For instance, in a synchronous semantics the choreography (2.4) requires that both sending and receiving actions on a happen before the ones on b. Given a synchronous semantics, such an ordering may be enforced in (2.4) if we have

$$\{\mathtt{s}_1, \mathtt{r}_1\} \cap \{\mathtt{s}_2, \mathtt{r}_2\} \neq \varnothing$$

- The *sender semantics* demands that the sequentiality of sending actions are preserved. In this case, the choreography (2.4) requires that the sending on a happens

before the sending on b, while receiving actions (by $\mathtt{r}_1$ and $\mathtt{r}_2$) may happen in any order. This may be enforced by $\mathtt{s}_2 = \mathtt{s}_1$ or $\mathtt{s}_2 = \mathtt{r}_1$.

- The *receiver semantics* is the dual of the sender semantics, it requires that receiving actions are ordered. This may be enforced in (2.4) by $\mathtt{s}_2 = \mathtt{r}_1$ or $\mathtt{r}_2 = \mathtt{r}_1$.

- The *disjoint semantics* requires the executions of two sequential interactions to be totally disjoint. In the choreography (2.4), this means that the sending on b must happen *after* $\mathtt{r}_1$ has received a. This may be enforced by $\mathtt{r}_1 = \mathtt{s}_2$.

For each of the semantics above, the authors formalise a notion of conformance, based on a bisimulation relation between POC and IOC. Then, for each notion of conformance, the authors give *connectedness* conditions which guarantee that a choreography and its projections are conformant.

Basically, the connectedness condition require that (*i*) there is a causality relation between sequential action in a IOC and (*ii*) that there is a unique point of choice.

- The causality relation requirement rules out IOC like

$$\mathtt{s}_1 \rightarrow \mathtt{r}_1 : \mathtt{a} \;\mid\; \mathtt{s}_2 \rightarrow \mathtt{r}_2 : \mathtt{a}$$

  which specifies two un-ordered interactions on the *same* operation. The corresponding POC of this choreography are as follows:

$$\mathtt{s}_1[!\mathtt{a}] \mid \mathtt{r}_2[?\mathtt{a}] \mid \mathtt{s}_2[!\mathtt{a}] \mid \mathtt{r}_2[?\mathtt{a}]$$

  which, according to the semantics of POC, would lead to a race condition, i.e., it may be the case that $\mathtt{s}_1$ interacts with $\mathtt{r}_2$ (instead of $\mathtt{s}_1$).

- The unique point of choice condition requires that each choice is made by exactly one participant, as in other theories such as [29, 40].

For each variations of the semantics, the authors show that if a choreography satisfies the connectedness conditions, then it is conformant with its projections (with respect to the semantics being considered).

Based on this work, [47] introduces a few techniques to amend choreographies so that the connectedness conditions hold.

Realisability. Bultan et al. [9, 10] tackle the problem of determining whether a choreography is realisable. Essentially, a choreography is realisable if "it is possible to build a distributed system that communicates exactly as the choreography specifies".

Choreographies in these works take the form of *conversation protocols*, that are finite state machines specifying the allowable sequence of interactions. A conversation protocol is akin to a generalised global type (cf. Section 2.1.4), except that parallel interactions cannot be explicitly represented, i.e., the ◫-gate of the generalised global types has no counterpart in conversation protocols. Analogously, the counterpart of local types (or POC) is specified in the form of (asynchronous) communicating machines, similar to the ones in [40] (cf. Section 2.1.4). A subtle difference with the communicating machines we presented earlier is that each machine has now a unique buffer from which it can receive messages. This is different from the machines presented previously where there are two buffers for each pair of machines, one in each direction.

In [9], the authors give necessary and sufficient conditions for deciding choreography realisability. More precisely, a choreography is realisable if there exists a system of communicating machines which generates exactly the set of message sequences (i.e., traces) specified by the choreography. A choreography is realisable if and only if:

- the language accepted by the conversation protocol is equivalent to the 1-bounded system of its projections, i.e., the transition system obtained by projecting the choreography on each participant and executing the system with the additional constraint that there is at most one symbol in each buffer, and

- the 1-bounded system satisfies a temporal logic property stating that every message sent by any participant must eventually be consumed by a receiver.

A consequence of this result is that checking for realisability may be done efficiently, namely, using off-the-shelf model checking tools.

2.2.2 On Synthesising Choreographies

To be the best of our knowledge, little work as been done with respect to synthesising a global view of a distributed system from local specifications. In this section, we discuss works which attempt to obtain a *formal* global descriptions from local specifications.

Synthesis of global types

In the session type setting, the most advanced result concerns the synthesis of global types from communicating machines. We discuss briefly an older work which first mentioned the synthesis of global types from local types.

From local types. A *bottom-up* approach to build choreographies is briefly studied in [57]. This work relies on global and local types, but uses *local and global graphs*. A local graph is similar to a local type while a global graph is a disjoint union of family of local graphs. In fact, global graphs do not explicitly give the structure of the interactions that the local types specify.

From communicating machines. In [41], Deniélou and Yoshida propose a characterisation of communicating machines, from which it is possible to synthesise a global type, and a synthesis algorithm.

They use local types generated from the following grammar:

$$P \;::=\; \mathtt{r}?\{\mathsf{a}_\mathsf{i};P_i\}_{i\in I} \;\mid\; \mathtt{s}!\{\mathsf{a}_\mathsf{i};P_i\}_{i\in I} \;\mid\; \mu\chi.P \;\mid\; \chi \;\mid\; \mathbf{0}$$

which may easily be translated to communicating machines (and vice versa). These communicating machines are called *basic* CFSMs and have the following properties

- they are deterministic, i.e., all the transitions leaving a same state have different labels,
- they have no mixed sates, i.e., all the transitions leaving a same state are either all receiving (`r?a`) or all sending (`r!a`), and
- they are directed, i.e., all the transitions leaving a same state are labelled by an action directed to the same machine.

The main contribution of this paper is the notion of *multiparty compatibility* which generalises the notion of duality between local types from [43] (cf. Section 2.1.1). Essentially, a system of CFSMs is multiparty compatible, if any action made by a machine can be "met" by the other machines, i.e., "the idea is to check the duality between each automaton and the rest, up to the internal communications (...)that the other machines will independently perform" [41]. Since CFSMs are generally undecidable [16], this check must be restricted to configurations of the system which are reachable by a 1-bounded execution, i.e., an execution of the system with the additional restriction that there is at most one symbol in each queue. Multiparty compatible systems enjoy the usual safety properties such as deadlock freedom, etc.

The algorithm to synthesise global types from multiparty compatible systems is quite simple and relies on a conversion from the transition system, corresponding to a synchronous execution of the CFSMs, to a global type. Remarkably, the global type so obtained is not unique and there is no special treatment of independent interactions, i.e., one may not synthesise a global type of the form $\mathcal{G} \mid \mathcal{G}'$. However, the projections of a synthesised global type are equivalent to the original system.

Other contributions of the authors in this paper include labelled transition system (LTS) semantics of both local types and global types, respectively.

Service composition

In [71, 72], Wang et al. give an algorithm to represent the composition of several services as a Petri net, such that an "optimal" representation is found. They consider three kinds of service models: input/output, precondition/effect and stateful models.

- In an *input/output model*, each operation of a service is represented by a pair of input and output data. For instance, a pair $(I_{\mathsf{a}}, O_{\mathsf{a}})$ says that, in order to execute operation a, the data I_{a} must have been previously generated (by other services or operations). After the execution of a, data O_{a} is generated.

- In a *precondition/effect model*, operations are defined as triples $(p_{\mathsf{a}}, E^+_{\mathsf{a}}, E^-_{\mathsf{a}})$. The element p_{a} is a set of propositions expressing the preconditions of the operation a: all the propositions must be true before executing operation a. The positive (resp. negative) effects E^+_{a} (resp. E^-_{a}) specify the propositions which are true (resp. false) after the execution of a.

- In a *stateful model*, a service is represented as an automaton, whose transitions represent operations.

Remarkably, the authors show that both input/output and precondition/effect models may be easily represented as automata. In particular, they build an automaton per operation and per variable used in a service.

In order to compose services (i.e., automata representing services), the authors use the parallel product of automata. The parallel product of two automata is a new automaton which, essentially, covers all possible "private" transitions of each original automaton, and the common transition are synchronised. Once the services have been combined into a unique automaton, the authors use the theory of regions [3] to synthesise a Petri net, using slight variations of well-established algorithms from [37].

One of the results presented by the authors is that the Petri net obtained from such a transformation is optimal in the sense that it is "maximally concurrent". However, in [71], no result shows that the Petri net obtained from the composition of services allows to recover the original services. In our terminology, it means that the authors do not guarantee that the global view of the services can be projected to a set of services equivalent to the original ones.

Inference of Message Sequence Charts.

In [1], Alur et al. focus on discovering whether Message Sequence Charts (MSC) [46, 54] imply unspecified scenarios. In fact, from a set of MSCs specifying what may seem to be correct scenarios, one may actually infer incorrect ones. This is due to the fact that, once implemented, each participant has only a local knowledge of the system. The authors assume that MSCs are implemented by concurrent automata, that are akin to the communicating machines we discussed earlier, but do not necessarily feature order-preserving communications.

The authors give precise conditions on MSCs for their implementation to be deadlock-free and realisable. Essentially, MSCs are realisable if no other MSC may be inferable from them, i.e., there is no other "implied" scenario. For the cases where MSCs are not realisable, the authors propose an algorithm to *synthesise* implied MSCs (i.e., unspecified scenarios which are in fact possible in the automata implementation).

Observe that the authors do not attempt to give an exhaustive global view of a distributed system, but instead focus on identifying "bad" executions. This is quite different from the approach we adopt in our work since we actually require that the projections of a synthesised choreography exhibits a behaviour that is equivalent to the original system.

2.2.3 Beyond Multiparty Session Types

In this section, we review a few remarkable extensions of the multiparty session types.

Dynamic session types. The original multiparty session types framework only caters for a fixed number of participants within a session. In [39], Deniélou and Yoshida introduced an extension of the framework to allow participants to join dynamically an already running session. To achieve this, the authors introduce a universal quantifier for *roles* in both global and local types; so that a (potentially unlimited) number of participants abiding by a common local type may join the session – provided that another participant exhibits a joining point.

Global programming. In [26], Carbone et al. introduce a novel methodology to distributed software design and implementation. They propose a language for global choreographies where instances of sessions are considered as first class element. In a sense this approach is a level of abstraction higher than global types – which describe only one session at a time. This permits to design and implement distributed systems from a global point of view. These "meta-choreographies" may then be projected onto end-point code. When some conditions holds for these global choreographies, the authors show that the implementation of the system, obtained by projection, is deadlock-free.

Global progress. One shortcoming of the original works on session types is that progress was only guaranteed within a single session. However, in realistic scenarios, a program may be involved in several sessions, possibly with different participants. In [11, 35, 36], the authors tackle the problem of *global progress*, i.e., ensuring that programs involved in several (interleaved) sessions do not deadlock. In Example 2.11 (adapted from [35]), we illustrate the paradigmatic example of a deadlock due to interleaved sessions.

Example 2.11. Consider the following processes:

$$R_{\mathtt{s}} \;=\; n_{1\,[\mathtt{s}]}(k_1)\,;\, n_{2\,[\mathtt{s}]}(k_2)\,;\, k_1\,?\,\mathtt{r}\,(\mathsf{a})\,;\, k_2\,!\,\mathtt{r}\langle \mathsf{b}\rangle$$

$$R_{\mathtt{r}} \;=\; \overline{n_1}_{[\mathtt{r}]}(k_1)\,;\, \overline{n_2}_{[\mathtt{r}]}(k_2)\,;\, k_2\,?\,\mathtt{s}\,(\mathsf{b})\,;\, k_1\,!\,\mathtt{s}\langle \mathsf{a}\rangle$$

where each process initiates two sessions on n_1 and n_2.

The parallel composition of these processes, $R_{\mathtt{s}} \mid R_{\mathtt{r}}$, deadlocks when it reduces to

$$(\nu k_1)\;(\nu k_2)\;\big(\; k_{1\,[\mathtt{s}]}\,?\,\mathtt{r}\,(\mathsf{a})\,;\, k_{2\,[\mathtt{s}]}\,!\,\mathtt{r}\langle \mathsf{b}\rangle \quad\mid\quad k_{2\,[\mathtt{r}]}\,?\,\mathtt{s}\,(\mathsf{b})\,;\, k_{1\,[\mathtt{r}]}\,!\,\mathtt{s}\langle \mathsf{a}\rangle\;\big)$$

where each process waits for a message from the other.

Notice that the program is well-typed in the sense of [44]. Indeed, assuming that the global type associated to n_1 is $\mathtt{r}\to\mathtt{s}:\mathsf{a}$ and that the global type associated to n_2 is $\mathtt{s}\to\mathtt{r}:\mathsf{b}$, both processes use each channel as prescribed by the projections of its corresponding global type. ◇

Intuitively, a process has the global progress property if every receive action will eventually be matched by a corresponding message and every sent message is eventually consumed. However, an important aspect considered in these works is that a process should not be considered stuck if the only reason why it cannot reduce further is because some participants are missing in order to initiate a new session. Indeed, in this case, the process will be able to proceed as soon as other participants join the system. In other words, global progress requires that, once a session has started, all the interactions that are supposed to occur in this session will eventually happen.

In order to guarantee global progress, the authors rely on a slight variation on the type system of the multiparty session types [44], which guarantees progress within a session. In addition, they introduce an *interaction type system* which, roughly, ensures that there is no circular dependencies between channel (or service) names.

2.2.4 Other Approaches to Multiparty Sessions

Many different approaches tackle the problem of designing and verifying formal models of distributed systems, which, to some extent, distance themselves from the framework based on global types. In this section, we discuss three approaches that are of particular interest in the scope of this thesis.

Compliance. In [31], Castagna et al. propose a *contract* language for processes. The language is akin to the π-calculus, allowing channel names to be passed around, and the semantics of contracts is synchronous.

In order to characterise when a set of contracts has progress, they adopt a testing approach [15] and define the notion of *compliance*. Essentially, a contract S is compliant with a contract S', if for any computation of their composition where S cannot progress by itself, either (*i*) the residual of S has terminated and S' will eventually terminate; or (*ii*) the two residuals (of S and S') can eventually synchronise. The notion of compliance is extended to the one of *well-formedness* of a system of contracts; which requires that each contract is compliant with rest of the system. Compliance is in fact similar to the notion of multiparty compatibility of [41].

Interestingly, the authors apply their approach to a calculus similar to the one of [11, 43]. They propose a type system for this calculus which extracts contracts from processes. Using the notion of well-formedness, they are able to give a progress result, stating that if a process is typable (i.e., by a system of contracts) and its type is "well-formed", then the process is able to reduce further or has terminated.

In the same line of work, [63] proposes to "project" processes into session types (akin to the contracts of [31] but without name passing capabilities). The author gives an adapted notion of compliance, now called *completeness* which permits to characterise when a set of session types is able to reduce further. Similarly to [31], session types may be extracted from processes (based on a variation of the π-calculus). If the types assigned

to a process are "complete", then progress and deadlock properties are guaranteed.

In addition, to ensure that channel delegation does not cause deadlock, the author defines a strict partial ordering on channels which is used to check whether a channel may be sent over another one. Interestingly, this allows channels to be used in a non-linear way, i.e, a process may send a channel to another process and still use this channel later.

Conversation types. Caires et al. [21, 23] propose a type-based framework to analyse and verify distributed systems. Their approach is somewhat similar to that of session types, in the sense that they propose a calculus (inspired from the π-calculus) and a type discipline for it. If a program is typable then it never violates the prescribed protocol and, under some condition, it is guaranteed to have progress. The framework allows for dynamical join and leave of participants, and *nested* conversations.

The framework is based on the conversation calculus [22, 69]. The main difference with respect to the approach of Honda et al. [44] is that interactions are passing through a *conversation*, i.e., a shared medium akin to a shared channel between several participants. A conversation may be dynamically created via a shared name – similarly to a session channels – and allows participants to dynamical join and leave a conversations by communicating the name of the conversation. A conversation may be typed by a *conversation type*. These behavioural types take into account both internal and external behaviours of participants in a conversation.

Contract oriented computing. In [5–7], Bartoletti et al. propose a novel approach, called contract-oriented computing, where the interactions between distributed entities are governed by contracts.

The life-cycle of contract-oriented computing consists of three phases. In the first phase, the entities' contracts are used to find an agreement where each participant's needs

and/or offers are matched by other contracts. In the second phase, the contracts become "binding", i.e., each entity will be deemed *culpable* if it cannot meet its obligations. In the third phase, the entities execute their tasks while monitoring their contracts.

The authors defined a new calculus, CO_2 (for COntract-Oriented computing) which may be parametrised with respect to a contract model. The calculus features basic primitives that permit to instantiate the contract-oriented paradigm, such as:

- a *tell* primitive to advertise a contract to a broker,
- a *fuse* primitive allowing a broker to create a new session between a set of participants who advertised compliant contracts, and
- a *do* primitive, used by participants to fulfil an action specified in their contracts.

An important feature of CO_2 is that it allows to identify participants who do not meet their requirement – so that, e.g., they may face legal consequences. The authors also define the notion of *honesty*, i.e., the ability of a participant to always be able to exculpate itself. In other words, an honest participant will always fulfil the contracts it advertised.

A negative result in [7] is that it is not decidable whether or no a given process is honest. In [7], CO_2 is instantiated to a theory of bilateral contracts inspired by [30]. A decidable approximation of honesty is introduced in [4], relying on the product between a finite state system (approximating contracts) and a Basic Parallel Process (approximating a CO_2 process).

Remark 2.6. *In Chapter 5, we define formally a slight variation of CO_2 where contracts are local session types and they are compliant when it is possible to synthesise a choreography from them. We also revisit the notion of honesty to our needs.*

CHAPTER 3

Synthesising Choreographies from Local Session Types

We introduce a type system which allows, under some conditions, to synthesise a choreography (i.e., a global type) from a set of local session types which describe end-point behaviours (i.e., local types). We show, notably, that the projections of a synthesised global type is equivalent to the original set of local types and that, if a set of local types is typable, then it enjoys safety and liveness properties.

3.1 Introduction

The multiparty session types framework proposes a methodology that (*i*) allows to design a global view of the interactions – aka *global type* –, (*ii*) provides an effective analysis of such a global view, (*iii*) automatically projects the global view to local end-points – aka *local types* –, and (*iv*) type checks end-points' code against local types. The theory guarantees that, when the global view enjoys suitable properties (phase (ii)), the end-points typable with local types enjoy, e.g., liveness properties such as progress.

A drawback of such an approach is that it cannot be applied when the local types describing the communication patterns of end-points are not obtained by an a priori designed global view. For instance, in service-oriented computing, one typically has independently

developed end-points that have to be combined to form larger services. Hence, deciding if the combined service respects its specification becomes non trivial. To illustrate this, we introduce a simple example used throughout this chapter.

Consider the system

$$S_{\mathrm{BS}} \;=\; \mathtt{b}_1[P_1] \;\mid \mathtt{s}_1[S_1] \;\mid \mathtt{b}_2[P_2] \;\mid \mathtt{s}_2[S_2]$$

consisting of two buyers ($\mathtt{b}_1$ and $\mathtt{b}_2$) and two sellers ($\mathtt{s}_1$ and $\mathtt{s}_2$) running in parallel, and whose behaviours are specified as follows.

$$\begin{aligned}
P_1 &= \mathtt{b}_2!\mathsf{agreement};\mathtt{b}_2?\mathsf{ack};Q_1 \\
P_2 &= \mathtt{b}_1?\mathsf{agreement};\mathtt{b}_1!\mathsf{ack};Q_2 \\
Q_{\mathtt{i}} &= \mathtt{s}_{\mathtt{i}}!\mathsf{request};\mathtt{s}_{\mathtt{i}}?\mathsf{quote};\mu\chi.(\mathtt{s}_{\mathtt{i}}!\mathsf{ok} \oplus \mathtt{s}_{\mathtt{i}}!\mathsf{no};\mathtt{s}_{\mathtt{i}}?\mathsf{quote};\chi) \qquad \mathtt{i} \in \{1,2\} \\
S_{\mathtt{i}} &= \mathtt{b}_{\mathtt{i}}?\mathsf{request};\mathtt{b}_{\mathtt{i}}!\mathsf{quote};\mu\chi'.(\mathtt{b}_{\mathtt{i}}?\mathsf{ok} + \mathtt{b}_{\mathtt{i}}?\mathsf{no};\mathtt{b}_{\mathtt{i}}!\mathsf{quote};\chi') \qquad \mathtt{i} \in \{1,2\}
\end{aligned}$$

where $\oplus$ represents an internal choice and $+$ a choice made by the environment. Intuitively, the overall behaviour of S_{BS} should be that two buyers ($\mathtt{b}_1$ and $\mathtt{b}_2$) agree on purchasing concurrently two items from their respective sellers and try to obtain the best price for these items. A natural question arises: is this intended behaviour actually realised by S_{BS}? Arguably, it is not immediate to answer such a question by considering only S_{BS}, even more so if a system involves a large number of participants, with possibly complex behaviours.

We propose to construct a global view of distributed end-points like S_{BS} via a slight variation of the multiparty session types introduced in [44]. Such types formalise a global view of the behaviour which, for S_{BS}, resembles the diagram in Figure 3.1, where the choreography of the overall protocol becomes much clearer.

An advantage of our approach is that we can reuse the results of the theory of multiparty session types to prove properties of end-points, e.g., safety and progress. In fact,

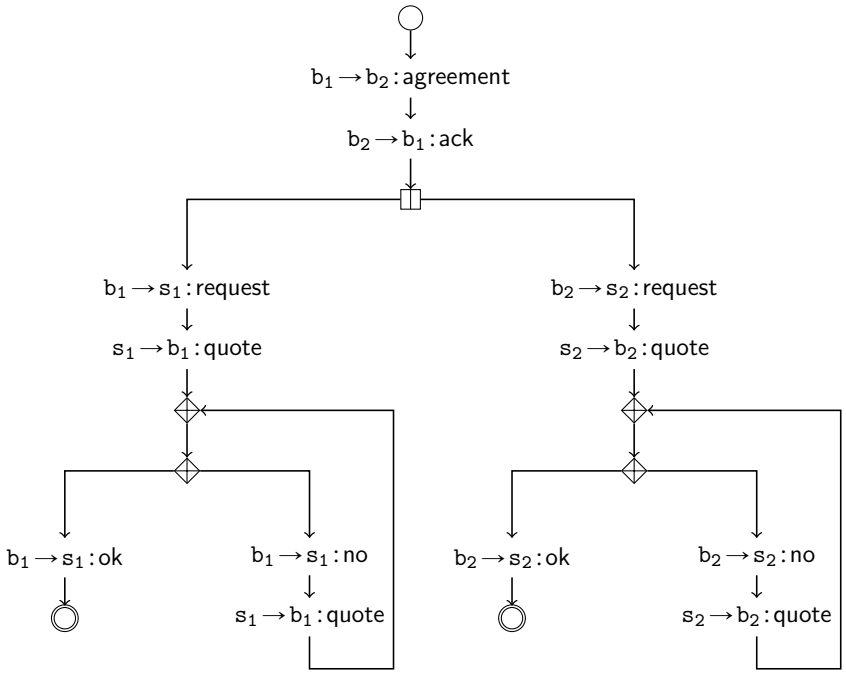


Figure 3.1: Global view of S_{BS}

we show that when the choreography can be constructed, its projections correspond to the initial end-points. Therefore, the well-formedness of the synthesised global choreography guarantees progress and safety properties of end-points.

We assume that session types are extracted from programs (relying on, e.g., [31, 43, 44]), and that they are readily available before addressing the construction of a global type.

3.2 Local Types

We use CCS-like processes (with guarded external and internal choices) to infer a global type from *local types* that correspond to the participants in the inferred choreography. Hereafter, $\mathbb{P}$ is a denumerable set of *participant names* (ranged over by $\mathtt{s}$, $\mathtt{r}$, $\mathtt{n}$, ...) and $\mathbb{C} = (\mathbb{P} \times \mathbb{P}) \backslash \{(\mathtt{n}, \mathtt{n}) \mid \mathtt{n} \in \mathbb{P}\}$ is a denumerable set of *channel identifiers*, we write $\mathtt{sr}$ when

$(\mathtt{s},\mathtt{r}) \in \mathbb{C}$.

Syntax. The syntax of local types below is parametrised with respect to basic data types such as $\mathsf{bool}, \mathsf{int}, \ldots$ (ranged over by a, b, c, ...):

$$\begin{array}{lcccccccc} S, S' & ::= & \mathtt{n}[P] & \mid & \mathtt{sr} : \rho & \mid & S \mid S' & \mid & \mathbf{0} \\ P, Q & ::= & \bigoplus_{i \in I} \mathtt{r}_i!\mathsf{a}_i; P_i & \mid & \sum_{i \in I} \mathtt{s}?\mathsf{a}_i; P_i & \mid & \mu\chi.P & \mid & \chi \\ \rho & ::= & \varepsilon & \mid & \mathsf{a} \cdot \rho & & & & \end{array}$$

A system S consists of the parallel composition of *processes* and *queues*. A *process* $\mathtt{n}[P]$ is a behaviour P identified by $\mathtt{n} \in \mathbb{P}$; we assume that, given a system S, the participant names are all different in S. A behaviour is either an internal choice, an external choice, or a recursive process. An internal choice $\bigoplus_{i \in I} \mathtt{r}_i!\mathsf{a}_i; P_i$ is guarded by output prefixes $\mathtt{r}_i!\mathsf{a}_i$ representing the sending of a value of sort a_i on the queue $\mathtt{sr}_i$, assuming the behaviour is the one of participant $\mathtt{s}$. We assume that

$$\forall i, j \in I : i \neq j \implies (\mathtt{r}_i, \mathsf{a}_i) \neq (\mathtt{r}_j, \mathsf{a}_j)$$

An external choice $\sum_{i \in I} \mathtt{s}?\mathsf{a}_i; P_i$ is guarded by input prefixes $\mathtt{s}?\mathsf{a}_i$ representing the reception of a value of sort a_i from the queue $\mathtt{sr}$, assuming the behaviour is the one of participant $\mathtt{r}$. We assume that

$$\forall i, j \in I : i \neq j \implies \mathsf{a}_i \neq \mathsf{a}_j$$

The asymmetry between internal and external choice, i.e., the fact that all branches of an external choice must be prefixed by an input from a same sender $\mathtt{s}$ allows us to guarantee that a choice is always made by exactly one participant (cf. Section 3.6).

We overload $\mathbf{0}$ to denote either an internal or external choice where $I = \varnothing$, i.e.,

$$\mathbf{0} \quad \stackrel{def}{=} \quad \bigoplus_{i \in \varnothing} \mathtt{r}_i!\mathtt{a}_i; P_i \quad = \quad \sum_{i \in \varnothing} \mathtt{s}?\mathtt{a}_i; P_i$$

In $\mu\chi.P$, $\mu\chi$ is a binder for the free occurrences of χ in P. Moreover, all such free occurrences are prefix-guarded in P. We consider closed behaviours only (that is, behaviours with no free occurrences of recursion variables). Also, for simplicity – and without loss of generality since bound variables are α-convertible – we assume that bound variables are pairwise distinct.

A *program* is a system with no queues, while a *runtime system* is a system S having exactly one queue $\mathtt{sr} : \rho$ per pair of participants $\mathtt{s}$ and $\mathtt{r}$ in S. We write $\mathtt{sr} : \varepsilon$ whenever the queue from $\mathtt{s}$ to $\mathtt{r}$ is empty. Given a program S, we write $\mathrm{Q}(S)$ for the parallel composition of the empty queues connecting all the participants in S. In the following, $S, S', \ldots$ denote either a program or runtime system.

Semantics. The semantics of systems is reminiscent of CCS with asynchronous and order-preserving communications. The semantics of systems is a labelled transition system (LTS) with labels:

$$\alpha \;::=\; \mathtt{r}!\mathtt{a} \;\mid\; \mathtt{s}?\mathtt{a}$$

$$\lambda \;::=\; \alpha \;\mid\; \mathtt{sr}\cdot\mathtt{a} \;\mid\; \mathtt{a}\cdot\mathtt{sr} \;\mid\; \mathtt{n}[\alpha] \;\mid\; \mathtt{s} \to \mathtt{r} : \mathtt{a} \;\mid\; \mathtt{r} \leftarrow \mathtt{s} : \mathtt{a}$$

Label α indicates either sending or reception by a process. Labels $\mathtt{sr}\cdot\mathtt{a}$ and $\mathtt{a}\cdot\mathtt{sr}$ respectively indicate push and pop operations on queues. Label $\mathtt{n}[\alpha]$ indicates a communication action done by participant $\mathtt{n}$. The next two labels indicate synchronisations between a participant and a queue: label $\mathtt{s} \to \mathtt{r} : \mathtt{a}$ indicates that $\mathtt{s}$ puts a datum on a queue $\mathtt{sr}$, while label $\mathtt{r} \leftarrow \mathtt{s} : \mathtt{a}$ indicates that $\mathtt{r}$ retrieves a datum from the queue $\mathtt{sr}$.

The semantics is given up to the congruence rules below.

$$\mu\chi.P \equiv P\{^{\mu\chi.P}/_{\chi}\}$$

$$S \mid S' \equiv S' \mid S \qquad S \mid \mathbf{0} \equiv S \qquad S' \mid (S' \mid S'') \equiv (S' \mid S') \mid S''$$

Note that commutativity and associativity of internal and external choice follow by construction (i.e., by the set notation used in the sums). We often write $P \oplus P'$ (resp. $P + P$) for internal (resp. external) choice. We assume that these operators are commutative and associative, and that $\_;\_$ has a higher precedence than $\_\oplus\_$ and $\_+\_$.

The LTS $\xrightarrow{\lambda}$ is the smallest relation closed under the following rules:

$$\text{[EXT]}\ \sum_{i\in I} \mathtt{s}?\mathtt{a}_i;P_i \xrightarrow{\mathtt{s}?\mathtt{a}_j} P_j \quad j \in I \qquad \text{[INT]}\ \bigoplus_{i\in I} \mathtt{r}_i!\mathtt{a}_i;P_i \xrightarrow{\mathtt{r}_j!\mathtt{a}_j} P_j \quad j \in I$$

$$\text{[BOX]}\ \frac{P \xrightarrow{\alpha} P'}{\mathtt{n}[P] \xrightarrow{\mathtt{n}[\alpha]} \mathtt{n}[P']} \qquad \text{[EQ-P]}\ \frac{P \equiv Q \xrightarrow{\alpha} Q' \equiv P'}{P \xrightarrow{\alpha} P'}$$

$$\text{[POP]}\ \mathtt{sr} : \mathtt{a} \cdot \rho \xrightarrow{\mathtt{a}\cdot\mathtt{sr}} \mathtt{sr} : \rho \qquad \text{[PUSH]}\ \mathtt{sr} : \rho \xrightarrow{\mathtt{sr}\cdot\mathtt{a}} \mathtt{sr} : \rho \cdot \mathtt{a}$$

$$\text{[IN]}\ \frac{S \xrightarrow{\mathtt{r}[\mathtt{s}?\mathtt{a}]} S_1 \qquad S' \xrightarrow{\mathtt{a}\cdot\mathtt{sr}} S_2}{S \mid S' \xrightarrow{\mathtt{r}\leftarrow\mathtt{s}:\mathtt{a}} S_1 \mid S_2} \qquad \text{[OUT]}\ \frac{S \xrightarrow{\mathtt{s}[\mathtt{r}!\mathtt{a}]} S_1 \qquad S' \xrightarrow{\mathtt{sr}\cdot\mathtt{a}} S_2}{S \mid S' \xrightarrow{\mathtt{s}\rightarrow\mathtt{r}:\mathtt{a}} S_1 \mid S_2}$$

$$\text{[EQ-S]}\ \frac{S \equiv S_1 \xrightarrow{\lambda} S_2 \equiv S'}{S \xrightarrow{\lambda} S'} \qquad \text{[PAR]}\ \frac{S \xrightarrow{\lambda} S'}{S \mid S'' \xrightarrow{\lambda} S' \mid S''}$$

Rules [EXT] and [INT] are trivial. By [POP], a queue sends the first datum, if any. By [PUSH], a queue receives a new datum. Processes can synchronise with queues according to rules [IN] and [OUT]. The remaining rules are standard. Let $S \rightarrow$ if and only if there are S' and λ such that $S \xrightarrow{\lambda} S'$ and $\xrightarrow{\lambda_1 \ldots \lambda_n}$ (resp. $\Longrightarrow$) be the reflexive transitive closure of $\xrightarrow{\lambda}$ (resp. $\rightarrow$).

We adopt the following notation.[1]

$$S(\mathtt{n}) = \begin{cases} P & \text{if } S \equiv \mathtt{n}[P] \mid S' \\ \bot & \text{otherwise} \end{cases} \qquad S[\mathtt{sr}] = \begin{cases} \rho & \text{if } S \equiv \mathtt{sr} : \rho \mid S' \\ \bot & \text{otherwise} \end{cases}$$

Given a system S and a participant name $\mathtt{n}$, $S(\mathtt{n})$ returns the behaviour of $\mathtt{n}$ in S, if $\mathtt{n}$ is in S; and is undefined otherwise. Similarly, $S[\mathtt{sr}]$ returns the content of the queue $\mathtt{sr}$, if the queue is part of S.

Example 3.1. Consider S_{BS} from the previous section, we have, e.g., $S_{\mathrm{BS}}(\mathtt{b}_1) = P_1$, while $S_{\mathrm{BS}}[\mathtt{b}_1\mathtt{b}_2] = \bot$ since S_{BS} is a program, i.e., it does not contain queues. ◇

3.3 Global Types

The global types we consider in this chapter are borrowed from [44] with minor variations. A global type is a term derivable from the following grammar:

$$\mathcal{G} \quad ::= \quad \textstyle\sum_{i\in I} \mathtt{s} \to \mathtt{r}_\mathtt{i} : \mathtt{a}_\mathtt{i}; \mathcal{G}_i \quad \mid \quad \mathcal{G} \mid \mathcal{G}' \quad \mid \quad \mu\chi.\mathcal{G} \quad \mid \quad \chi \quad \mid \quad \mathbf{0}$$

An expression of the form $\mathtt{s} \to \mathtt{r} : \mathtt{a}$ represents an interaction where $\mathtt{s} \in \mathbb{P}$ sends a value of sort $\mathtt{a}$ to $\mathtt{r} \in \mathbb{P}$ (we assume that $\mathtt{s} \neq \mathtt{r}$). The choice production $\sum_{i\in I} \mathtt{s} \to \mathtt{r}_\mathtt{i} : \mathtt{a}_\mathtt{i}; \mathcal{G}_i$ is guarded by prefixes $\mathtt{s} \to \mathtt{r}_\mathtt{i} : \mathtt{a}_\mathtt{i}$. The production indicates a (exclusive) choice of interactions, if participant $\mathtt{s}$ sends a message $\mathtt{a}_\mathtt{i}$ to $\mathtt{r}_\mathtt{i}$, then interactions $\mathcal{G}_i$ take place; we assume that

$$\forall i, j \in I : \; i \neq j \implies (\mathtt{r}_\mathtt{i}, \mathtt{a}_\mathtt{i}) \neq (\mathtt{r}_\mathtt{j}, \mathtt{a}_\mathtt{j})$$

We often write $\mathcal{G} + \mathcal{G}'$ for the binary version of the choice production, we assume that $\_ + \_$ is commutative and associative, and that the operator $\_;\_$ has a higher precedence

[1] We write $f(x) = \bot$ when the function f is undefined on x.

than $\_ + \_$. Concurrent interactions are written $\mathcal{G} \mid \mathcal{G}'$ and indicate sets of independent interactions. In a recursive global type $\mu\chi.\mathcal{G}$, χ is bound and guarded in $\mathcal{G}$. We assume that global types are closed and often omit trailing occurrences of $\mathbf{0}$.

Remark 3.1. *The main difference with respect to the global types in [44] is that we merge the branching constructs with the interaction prefix, similarly to [28], and we do not require a choice construct to involve only two participants, e.g., we accept the following global type:*

$$\mathtt{s} \to \mathtt{r}_1 : \mathtt{a}; \mathtt{s} \to \mathtt{r}_2 : \mathtt{a} \quad + \quad \mathtt{s} \to \mathtt{r}_2 : \mathtt{b}; \mathtt{s} \to \mathtt{r}_1 : \mathtt{b}$$

Global types are taken up-to structural congruence, defined as the smallest equivalence relation satisfying the monoidal axioms for $\_ \mid \_$ and $\_ + \_$ (with $\mathbf{0}$ as the identity element); and the axiom

$$\mu\chi.\mathcal{G} \equiv \mathcal{G}\{^{\mu\chi.\mathcal{G}}/_{\chi}\}$$

Example 3.2. The first two interactions between $\mathtt{b}_1$ and $\mathtt{b}_2$ in the example of Section 3.1 are modelled by the global type $\mathcal{G}_1$:

$$\mathcal{G}_1 \;=\; \mathtt{b}_1 \to \mathtt{b}_2 : \mathsf{agreement}; \mathtt{b}_2 \to \mathtt{b}_1 : \mathsf{ack}$$

The type $\mathcal{G}_1$ says that participant $\mathtt{b}_1$ sends a message of sort agreement to $\mathtt{b}_2$, then $\mathtt{b}_2$ replies with a message of sort ack.

Each concurrent branch of the interactions in Figure 3.1 are specified by the following global type (where $\mathtt{i} \in \{1,2\}$)

$$\begin{aligned} \mathcal{G}_2^{\mathtt{i}} \;=\;& \mathtt{b}_{\mathtt{i}} \to \mathtt{s}_{\mathtt{i}} : \mathsf{request}; \mathtt{s}_{\mathtt{i}} \to \mathtt{b}_{\mathtt{i}} : \mathsf{quote}; \\ & \mu\chi.(\mathtt{b}_{\mathtt{i}} \to \mathtt{s}_{\mathtt{i}} : \mathsf{ok} \;+\; \mathtt{b}_{\mathtt{i}} \to \mathtt{s}_{\mathtt{i}} : \mathsf{no}; \mathtt{s}_{\mathtt{i}} \to \mathtt{b}_{\mathtt{i}} : \mathsf{quote}; \chi) \end{aligned}$$

The global type $\mathcal{G}_2^{\mathtt{i}}$ says that $\mathtt{b}_{\mathtt{i}}$ sends a request to $\mathtt{s}_{\mathtt{i}}$, who replies with a message of type quote. Then, the two participants enter a recursion which is guarded by a choice made by

$\mathtt{b_i}$. If $\mathtt{b_i}$ is satisfied with the quote, the interactions terminate here; otherwise $\mathtt{s_i}$ sends another quote and the interactions are repeated.

Finally, the global type representing all the interactions in Figure 3.1 is given below.

$$\mathcal{G}_{S_{\mathrm{BS}}} \;=\; \mathtt{b_1} \to \mathtt{b_2} : \mathsf{agreement};\, \mathtt{b_2} \to \mathtt{b_1} : \mathsf{ack};\, (\mathcal{G}_2^1 \mid \mathcal{G}_2^2)$$

$\diamond$

The syntax of global types may specify behaviours that are not implementable. The rest of this section borrows from [28] and [44] and adapts the requirements a global type must fulfil to ensure that the set of interactions it prescribes is indeed feasible.

3.3.1 Well-formed Global Types

We define the conditions for a global type to be well-formed. We write $\mathcal{P}(\mathcal{G})$ (resp. $\mathcal{C}(\mathcal{G})$) for the set of participant (resp. channel) names in $\mathcal{G}$, and $\mathtt{fv}(\mathcal{G})$ (resp. $\mathtt{bv}(\mathcal{G})$) for the set of free (resp. bound) variables in $\mathcal{G}$, similarly for a system S.

We give a few auxiliary functions. The *ready set* of a global type $\mathcal{G}$, written $\mathtt{R}(\mathcal{G})$, is defined as follows.

$$\mathtt{R}(\mathcal{G}) \stackrel{def}{=} \{\mathtt{s} \to \mathtt{r} : \mathtt{a} \mid \mathcal{G} \equiv (\mathtt{s} \to \mathtt{r} : \mathtt{a};\, \mathcal{G}_1 + \mathcal{G}_2) \mid \mathcal{G}_3\}$$

Next we define a functions that computes the sets of participant which interact inde-

pendently "in the last part of a global type". We define $\mathtt{Indep}(\mathcal{G}) \stackrel{def}{=} \mathtt{I_0}(\varnothing, \mathcal{G})$, where

$$\mathtt{I_0}(\mathcal{P}, \mathcal{G}) \stackrel{def}{=} \begin{cases} \mathtt{I_0}(\{\mathtt{s},\mathtt{r}\} \cup \mathcal{P}, \mathcal{G}_1), & \text{if } \mathcal{G} = \mathtt{s} \to \mathtt{r} : \mathtt{a}; \mathcal{G}_1 \\ \mathtt{I_0}(\varnothing, \mathcal{G}_1) \cup \mathtt{I_0}(\varnothing, \mathcal{G}_2), & \text{if } \mathcal{G} = \mathcal{G}_1 \mid \mathcal{G}_2 \\ \mathtt{I_0}(\mathcal{P}, \mathcal{G}_1), & \text{if } \mathcal{G} = \mathcal{G}_1 + \mathcal{G}_2 \text{ and } \mathtt{I_0}(\mathcal{P}, \mathcal{G}_1) = \mathtt{I_0}(\mathcal{P}, \mathcal{G}_2) \\ \mathtt{I_0}(\mathcal{P}, \mathcal{G}_1), & \text{if } \mathcal{G} = \mu\chi.\mathcal{G}_1 \\ \{\mathcal{P}\}, & \text{if } \mathcal{G} = \mathbf{0} \text{ or } \mathcal{G} = \chi \\ \bot, & \text{otherwise} \end{cases}$$

$\mathtt{I_0}(\mathcal{P}, \mathcal{G})$ is the family of sets of participants of $\mathcal{G}$, so that for all $N \neq M \in \mathtt{I_0}(\mathcal{P}, \mathcal{G})$, the participants in N and those in M are in different concurrent branches "in the last part of $\mathcal{G}$". Note that $\mathtt{I_0}(\_,\_)$ is a partial function.

Example 3.3. Continuing with the global types from Example 3.2, we have:

$$\mathtt{R}(\mathcal{G}_{S_{\mathrm{BS}}}) = \mathtt{R}(\mathcal{G}_1) = \{\mathtt{b}_1 \to \mathtt{b}_2 : \mathsf{agreement}\} \quad \text{and} \quad \mathtt{R}(\mathcal{G}_2^1) = \{\mathtt{b}_1 \to \mathtt{s}_1 : \mathsf{request}\}$$

$$\mathtt{Indep}(\mathcal{G}_{S_{\mathrm{BS}}}) = \{\{\mathtt{b}_1, \mathtt{s}_1\}, \{\mathtt{b}_2, \mathtt{s}_2\}\}, \quad \mathtt{Indep}(\mathcal{G}_1) = \{\{\mathtt{b}_1, \mathtt{b}_2\}\}, \text{ and}$$

$$\mathtt{Indep}(\mathcal{G}_2^1) = \{\{\mathtt{b}_1, \mathtt{s}_1\}\}$$

◇

We say that $\mathcal{G}$ is *well-formed* if $\vdash \mathcal{G}$ can be derived from the rules given in Figure 3.2. We assume that each premise of the rules in Figure 3.2 does not hold whenever any of the auxiliary function used is undefined (e.g., in [WF-$\mu\chi$], if $\mathtt{Indep}(\mathcal{G}) = \bot$ then $\vdash \mu\chi.\mathcal{G}$ is not derivable). In the following, we discuss the rules of Figure 3.2, which are grouped according to three requirements: sequentiality, single threadness, and knowledge of choice.

Sequentiality [28]. Rule [WF-;] ensures that sequentiality is preserved. In [WF-;], the ordering dependency between a prefix and its continuation allows us to implement each

$$[\text{WF-};]\ \frac{\forall\, \mathtt{s}' \to \mathtt{r}' : \_ \in \mathtt{R}(\mathcal{G}) \,:\, \{\mathtt{s}', \mathtt{r}'\} \cap \{\mathtt{s}, \mathtt{r}\} \neq \varnothing \qquad \vdash \mathcal{G}}{\vdash \mathtt{s} \to \mathtt{r} : \mathtt{a}; \mathcal{G}}$$

$$[\text{WF-}|]\ \frac{\mathcal{P}(\mathcal{G}) \cap \mathcal{P}(\mathcal{G}') = \varnothing \qquad \vdash \mathcal{G} \qquad \vdash \mathcal{G}'}{\vdash \mathcal{G} \mid \mathcal{G}'}$$

$$[\text{WF-}\mu\chi]\ \frac{\chi \in \mathtt{fv}(\mathcal{G}) \Rightarrow |\mathtt{Indep}(\mathcal{G})| = 1 \qquad \vdash \mathcal{G}}{\vdash \mu\chi.\mathcal{G}}$$

$$[\text{WF-}\chi]\ \frac{}{\vdash \chi} \qquad\qquad [\text{WF-}\mathbf{0}]\ \frac{}{\vdash \mathbf{0}}$$

$$[\text{WF-}+]\ \frac{\begin{array}{c}\forall\, \mathtt{s} \to \mathtt{r} : \mathtt{a} \in \mathtt{R}(\mathcal{G}) :\ \forall\, \mathtt{s}' \to \mathtt{r}' : \mathtt{a}' \in \mathtt{R}(\mathcal{G}') :\ \mathtt{s} = \mathtt{s}' \wedge (\mathtt{r}, \mathtt{a}) \neq (\mathtt{r}', \mathtt{a}') \\ \vdash \mathcal{G} \qquad \vdash \mathcal{G}'\end{array}}{\vdash \mathcal{G} + \mathcal{G}'}$$

Figure 3.2: Rules for well-formedness

participant so that at least one action of the first prefix always happens before an action of the second prefix. More concretely, this rules out, e.g.,

$$\mathtt{s}_1 \to \mathtt{r}_1 : \mathtt{a}; \mathtt{s}_2 \to \mathtt{r}_2 : \mathtt{b} \qquad \text{✘}$$

where, evidently, it is not possible to guarantee that $\mathtt{s}_2$ sends $\mathtt{b}$ after $\mathtt{r}_1$ receives $\mathtt{a}$ from $\mathtt{s}_1$. Since we are working in an asynchronous setting, we do not want to force both send and receive actions of the first prefix to happen before both actions of the second one.

Single threadness [44]. A participant should not appear in different concurrent branches of a global type, so that each participant is single threaded. This is also reflected in the calculus of Section 3.2, where parallel composition is only allowed at the system level. Therefore, in [WF-|], the participant names in concurrent branches must be disjoint. Rule [WF-$\mu\chi$] requires that $\mathcal{G}$ is single threaded, i.e., concurrent branches cannot appear under recursion. If that was the case, a participant could appear in different concurrent branches of the unfolding of a recursive global type. For instance, the one-time unfolding of $\mathcal{G}$

below is $\mathcal{G}'$.

$$\mathcal{G} = \mu\chi.\big(\mathtt{s}_1 \to \mathtt{r}_1 : \mathtt{a};\, \chi \mid \mathtt{s}_2 \to \mathtt{r}_2 : \mathtt{b};\, \chi\big) \qquad \mathcal{G}' = \big(\mathtt{s}_1 \to \mathtt{r}_1 : \mathtt{a};\, \mathcal{G}\big) \mid \big(\mathtt{s}_2 \to \mathtt{r}_2 : \mathtt{b};\, \mathcal{G}\big) \qquad \text{✘}$$

Observe that the sets of participants in the concurrent branches of $\mathcal{G}'$ are not disjoint.

Knowledge of choice [28, 44]. Whenever a global type specifies a choice of two sets of interactions, the decision should be made by exactly one participant. For instance,

$$\mathtt{s}_1 \to \mathtt{r}_1 : \mathtt{a};\, \mathcal{G}_1 \quad + \quad \mathtt{s}_2 \to \mathtt{r}_2 : \mathtt{b};\, \mathcal{G}_2 \qquad \text{✘}$$

specifies a choice made by $\mathtt{s}_1$ in the first branch and by $\mathtt{s}_2$ in the second one; this kind of choreographies cannot be implemented (without using hidden interactions). Also, we want to avoid global types where a participant $\mathtt{n}$ behaves differently in choice branches without being aware of the choice made by others. For instance, in

$$\mathtt{s} \to \mathtt{r} : \mathtt{a};\, \mathtt{n} \to \mathtt{r} : \mathtt{c};\, \mathcal{G}_1 \quad + \quad \mathtt{s} \to \mathtt{r} : \mathtt{b};\, \mathtt{n} \to \mathtt{r} : \mathtt{d};\, \mathcal{G}_2 \qquad \text{✘}$$

where $\mathtt{n}$ ignores the choice of $\mathtt{s}$ and behaves differently in each branch. On the other hand, we want global types of the following form to be accepted.

$$\begin{array}{l} \mathtt{s} \to \mathtt{r} : \mathtt{a};\, \mathtt{n} \to \mathtt{s} : \mathtt{b};\, \mathtt{s} \to \mathtt{n} : \mathtt{a};\, \mathtt{n} \to \mathtt{r} : \mathtt{a} \\ \qquad\qquad + \qquad\qquad \text{✔} \\ \mathtt{s} \to \mathtt{r} : \mathtt{b};\, \mathtt{n} \to \mathtt{s} : \mathtt{b};\, \mathtt{s} \to \mathtt{n} : \mathtt{c};\, \mathtt{n} \to \mathtt{r} : \mathtt{d} \end{array}$$

Indeed, in this case $\mathtt{n}$ behaves differently in each branch, but only *after* "being informed" by $\mathtt{s}$ about the chosen branch, i.e., $\mathtt{n}$ always sends $\mathtt{b}$, then if it receives $\mathtt{a}$, it sends $\mathtt{a}$ again; if it receives $\mathtt{c}$, then it sends $\mathtt{d}$.

Together with the definitions of the projections of a global type (cf. Definition 3.1

below) rule [WF-+] guarantees that "knowledge of choice" is respected. In particular, the rule requires that the participant who makes the decision is the same in every branch of a choice, while the prefixes guarding the choice must be distinct.

Definition 3.1 ($\_\upharpoonright\_$)**.** *The projection of a global type $\mathcal{G}$ with respect to a participant* $\mathtt{n}$ *– written $\mathcal{G}\upharpoonright_{\mathtt{n}}$ – is defined by the (partial) function below.*

$$\mathcal{G}\upharpoonright_{\mathtt{n}} \stackrel{def}{=} \begin{cases} \mathtt{s}?\mathtt{a};\mathcal{G}'\upharpoonright_{\mathtt{n}}, & \text{if } \mathcal{G} = \mathtt{s}\to\mathtt{n}:\mathtt{a};\mathcal{G}' \\ \mathtt{r}!\mathtt{a};\mathcal{G}'\upharpoonright_{\mathtt{n}}, & \text{if } \mathcal{G} = \mathtt{n}\to\mathtt{r}:\mathtt{a};\mathcal{G}' \\ \mathcal{G}'\upharpoonright_{\mathtt{n}}, & \text{if } \mathcal{G} = \mathtt{s}\to\mathtt{r}:\mathtt{a};\mathcal{G}' \text{ and } \mathtt{s}\neq\mathtt{n}\neq\mathtt{r} \\ \mathcal{G}_1\upharpoonright_{\mathtt{n}} \oplus \mathcal{G}_2\upharpoonright_{\mathtt{n}}, & \text{if } \mathcal{G} = \mathcal{G}_1 + \mathcal{G}_2 \text{ and } \exists\, \mathtt{n}\to\mathtt{r}:\_ \in \mathtt{R}(\mathcal{G}) \\ \mathcal{G}_1\upharpoonright_{\mathtt{n}} \uplus \mathcal{G}_2\upharpoonright_{\mathtt{n}}, & \text{if } \mathcal{G} = \mathcal{G}_1 + \mathcal{G}_2 \text{ and } \forall\, \mathtt{s}\to\mathtt{r}:\_ \in \mathtt{R}(\mathcal{G}) : \mathtt{n}\neq\mathtt{s} \\ \mathcal{G}_i\upharpoonright_{\mathtt{n}}, & \text{if } \mathcal{G} = \mathcal{G}_1 \mid \mathcal{G}_2 \text{ and } \mathtt{n}\notin\mathcal{P}(\mathcal{G}_j),\ i\neq j\in\{1,2\} \\ \mu\chi.\mathcal{G}'\upharpoonright_{\mathtt{n}}, & \text{if } \mathcal{G} = \mu\chi.\mathcal{G}' \text{ and } \mathcal{G}'\upharpoonright_{\mathtt{n}} \neq \chi \\ \mathbf{0}, & \text{if } \mathcal{G} = \mu\chi.\mathcal{G}' \text{ and } \mathcal{G}'\upharpoonright_{\mathtt{n}} = \chi \\ \mathcal{G}, & \text{if } \mathcal{G} = \chi \text{ or } \mathcal{G} = \mathbf{0} \\ \bot, & \text{otherwise} \end{cases}$$

We say that a global type is projectable *if $\mathcal{G}\upharpoonright_{\mathtt{n}}$ is defined for all $\mathtt{n}\in\mathcal{P}(\mathcal{G})$.*

The projection map is similar to the one given in [44], but for the use of $\_\uplus\_$ to project choice branches (see Definition 3.2 below) – a similar operation is also used in [39].

The first three cases deal with prefixes and are trivial. The fourth case projects choice branches when the considered participant is the sender. Note that if $\mathcal{G}$ is well-formed then there is always a single sender participant in choice branches. The fifth case deals with branches when the considered participant is not the sender, in which cases the branches must be merged (see Definition 3.2 below). Concurrent branches may be projected only if

their sets of participants are disjoint. There are two cases to project recursive global types. The first case is used to project the behaviour of participants who are indeed involved in the recursion, while the second case deals with participants who do not appear after the recursion definition. The other cases are trivial.

Note that a global type may be projected even if it is not well-formed, but in that case none of the properties given below are guaranteed to hold.

Essentially, the function $\_ \uplus \_$ merges (if possible) the behaviour of a participant in different choice branches, cf. Example 3.4 below.

Definition 3.2 ($\_ \uplus \_$)**.**

$$P \uplus Q = \begin{cases} P + Q, & \text{if } P = \sum_{i \in I} \mathtt{s}?\mathtt{a}_i\,;P_i \text{ and } Q = \sum_{j \in J} \mathtt{s}?\mathtt{a}_j\,;Q_j \\ & \text{and } \forall i \in I:\ \forall j \in J:\ \mathtt{a}_i \neq \mathtt{a}_j \text{ and } I, J \neq \varnothing \\ \sum_{i \in I} \mathtt{s}?\mathtt{a}_i\,;P_i \uplus P'_i, & \text{if } P = \sum_{i \in I} \mathtt{s}?\mathtt{a}_i\,;P_i \text{ and } Q = \sum_{i \in I} \mathtt{s}?\mathtt{a}_i\,;P'_i \\ \bigoplus_{i \in I} \mathtt{r}_i!\mathtt{a}_i\,;P_i \uplus P'_i, & \text{if } P = \bigoplus_{i \in I} \mathtt{r}_i!\mathtt{a}_i\,;P_i \text{ and } Q = \bigoplus_{i \in I} \mathtt{r}_i!\mathtt{a}_i\,;P'_i \\ P, & \text{if } P \equiv Q \\ \bot, & \text{otherwise} \end{cases}$$

In the first case, the function merges two guarded external choices, if the message sorts in their prefix are disjoint. This ensures that each participant knows which branch of the global type was chosen. In the second and third cases, the function merges the continuation of both processes, if both have the same prefix. This allows for participants to have a common behaviour in both branches until they have enough information to know which branch was chosen. The fourth case deals with process of the form $\mathbf{0}$ and χ.

Note that $\_ \uplus \_$ is a partial function, e.g., it might be the case that a participant behaves differently in two branches without being aware of which branch was chosen, in such a

case the projection of that participant is undefined, cf. the last case of the definition.

Example 3.4. We illustrate the use of $\_ \uplus \_$ in a projection. Consider the global type

$$\mathcal{G} \;=\; (\mathtt{s} \to \mathtt{r} : \mathsf{a};\, \mathcal{G}_3) + (\mathtt{s} \to \mathtt{r} : \mathsf{b};\, \mathcal{G}_3)$$

such that $\mathcal{G}_3 = \mathtt{s} \to \mathtt{r}' : \mathsf{a} + \mathtt{s} \to \mathtt{r}' : \mathsf{b}$, and $\mathsf{a} \neq \mathsf{b}$.

- For $\mathtt{s}$, we have $\mathcal{G}_3 \upharpoonright_{\mathtt{s}} = \mathtt{r}'!\mathsf{a} \oplus \mathtt{r}'!\mathsf{b}$ and $\mathcal{G} \upharpoonright_{\mathtt{s}} = (\mathtt{r}!\mathsf{a};\, \mathcal{G}_3 \upharpoonright_{\mathtt{s}}) \oplus (\mathtt{r}!\mathsf{b};\, \mathcal{G}_3 \upharpoonright_{\mathtt{s}})$

  The projection onto the participant that makes the choice is simple since, by well-formedness, the sender is the same in each branch, and the guards are pairwise distinct.

- For $\mathtt{r}$, we have $\mathcal{G}_3 \upharpoonright_{\mathtt{r}} = \mathbf{0}$, since $\mathtt{r}$ does not appear in $\mathcal{G}_3$, and

$$\mathcal{G} \upharpoonright_{\mathtt{r}} \;=\; \mathtt{s}?\mathsf{a};\, \mathcal{G}_3 \upharpoonright_{\mathtt{r}} \uplus \mathtt{s}?\mathsf{b};\, \mathcal{G}_3 \upharpoonright_{\mathtt{r}} \;=\; \mathtt{s}?\mathsf{a} + \mathtt{s}?\mathsf{b}$$

  i.e., $\_ \uplus \_$ is defined here since $\mathsf{a} \neq \mathsf{b}$.

- For $\mathtt{r}'$, we have $\mathcal{G}_3 \upharpoonright_{\mathtt{r}'} = \mathtt{s}?\mathsf{a} \uplus \mathtt{s}?\mathsf{b}$, which is defined since $\mathsf{a} \neq \mathsf{b}$; and we have $\mathcal{G} \upharpoonright_{\mathtt{r}'} = \mathcal{G}_3 \upharpoonright_{\mathtt{r}'} \uplus \mathcal{G}_3 \upharpoonright_{\mathtt{r}'}$, which is defined since the arguments of $\uplus$ are the same.

$\diamond$

For the sake of readability, we have defined the projection of a global type using the binary operator $\_ + \_$. In fact, the projection of a global type is sensitive to the way choice branches are associated, as we show in Example 3.5 below.

Example 3.5. Consider the global type below.

$$\begin{array}{rcll} \mathcal{G}_{\text{ex3.5}} & = & \mathtt{s} \to \mathtt{r} : \mathtt{a};\ \mathtt{s} \to \mathtt{r}' : \mathtt{b} & (1) \\ & & + & \\ & & \mathtt{s} \to \mathtt{r}' : \mathtt{b};\ \mathtt{s} \to \mathtt{r} : \mathtt{a} & (2) \\ & & + & \\ & & \mathtt{s} \to \mathtt{r} : \mathtt{c};\ \mathtt{s} \to \mathtt{r}' : \mathtt{d} & (3) \end{array}$$

The success of the projection of $\mathcal{G}_{\text{ex3.5}}$ onto participant $\mathtt{r}$ (and $\mathtt{r}'$) depends on how the branches of the global type are partitioned. Let us project each branch onto $\mathtt{r}$ separately:

$$(1) : \ \mathtt{s?a} \qquad (2) : \ \mathtt{s?a} \qquad (3) : \ \mathtt{s?c}$$

If $\_ \uplus \_$ is used to merge branches (1) and (2) first, we obtain $\mathtt{s?a}$ which may be merged with branch (3), i.e., $\mathtt{s?a} \uplus \mathtt{s?c} = \mathtt{s?a} + \mathtt{s?c} \neq \bot$.

However, if we first merge branches (2) and (3), we obtain $\mathtt{s?a} + \mathtt{s?c}$, which may not be merged with branch (1), i.e., $(\mathtt{s?a} + \mathtt{s?c}) \ \uplus \ \mathtt{s?a} = \bot$. $\diamond$

In order to cater for global types such as $\mathcal{G}_{\text{ex3.5}}$ above, we introduce a more precise version of the case of Definition 3.1 dealing with the projection of choice branches onto a *receiver* participant.

Let $\mathcal{G} = \sum_{i \in I} \mathtt{s} \to \mathtt{r_i} : \mathtt{a_i};\ \mathcal{G}_i$, the projection of $\mathcal{G}\upharpoonright_{\mathtt{n}}$, with $\mathtt{n} \in \{\mathtt{r_i} \mid i \in I\}$, is defined as $\mathcal{G}\upharpoonright_{\mathtt{n}} \stackrel{def}{=} \mathcal{G}_1\upharpoonright_{\mathtt{n}} \uplus \mathcal{G}_2\upharpoonright_{\mathtt{n}}$, if the following holds:

$$\begin{array}{ll} \exists J, K :\ J \cup K = I \text{ and } J \cap K = \varnothing \text{ such that} & \mathcal{G}_1 = \sum_{j \in J} \mathtt{s} \to \mathtt{r_j} : \mathtt{a_j};\ \mathcal{G}_j, \\ & \mathcal{G}_2 = \sum_{k \in K} \mathtt{s} \to \mathtt{r_k} : \mathtt{a_k};\ \mathcal{G}_k, \text{and} \\ & \mathcal{G}_1\upharpoonright_{\mathtt{n}} \uplus \mathcal{G}_2\upharpoonright_{\mathtt{n}} \neq \bot \end{array}$$

Note that because $\_ \uplus \_$ is not used to project choice branches onto the participant who

makes the choice, there is no need to cater for associativity in that case. In addition, if the projection of a global type onto a participant is defined, then it is unique, cf. Lemma 3.1 below.

3.3.2 Properties of Well-formed Global Types

We give the basic properties which hold for well-formed global types. Some of these properties are based on previous results from [40], thus we show that these results are applicable in our setting.

Lemma 3.1. *For all $\mathcal{G}$ such that $\vdash \mathcal{G}$ and $\mathcal{G}$ is projectable, let $S = \big|_{\mathtt{n}\in\mathcal{P}(\mathcal{G})} \mathtt{n}[\mathcal{G}\upharpoonright_{\mathtt{n}}]$, the following holds:*

1. $\forall\, \mathtt{n} \in \mathcal{P}(\mathcal{G}) :\ \mathcal{G}\upharpoonright_{\mathtt{n}}$ *is unique.*
2. $\forall\, \mathtt{n} \in \mathcal{P}(\mathcal{G}) :\ \mathcal{G}\upharpoonright_{\mathtt{n}}$ *is a process.*
3. *S is a program.*
4. *$S \mid \mathrm{Q}(S)$ is a runtime system.*

Proof. Item 1 follows from the condition that sets of participants are pairwise disjoint in $\mathcal{G} \mid \mathcal{G}'$, and commutativity and associativity of $+$. We show the latter by contradiction. Assume we have $\mathcal{G} = (\mathcal{G}_1 + \mathcal{G}_2) + \mathcal{G}_3$ and $\mathcal{G}' = \mathcal{G}_1 + (\mathcal{G}_2 + \mathcal{G}_3)$ with $\exists\, \mathtt{n} \in \mathcal{P}(\mathcal{G})$ such that $\mathcal{G}\upharpoonright_{\mathtt{n}} \neq \mathcal{G}'\upharpoonright_{\mathtt{n}}$, $\mathcal{G}\upharpoonright_{\mathtt{n}} \neq \bot$, and $\mathcal{G}'\upharpoonright_{\mathtt{n}} \neq \bot$.

- If $\mathtt{n}$ is the sender, then, by Definition 3.1, we have

$$\mathcal{G}\upharpoonright_{\mathtt{n}} = (\mathcal{G}_1\upharpoonright_{\mathtt{n}} \oplus \mathcal{G}_2\upharpoonright_{\mathtt{n}}) \oplus \mathcal{G}_3\upharpoonright_{\mathtt{n}} \quad \text{and} \quad \mathcal{G}\upharpoonright_{\mathtt{n}} = \mathcal{G}_1\upharpoonright_{\mathtt{n}} \oplus (\mathcal{G}_2\upharpoonright_{\mathtt{n}} \oplus \mathcal{G}_3\upharpoonright_{\mathtt{n}})$$

  and the result follows directly by definition of processes (i.e., by associativity of $\_ \oplus \_$).

- If $\mathtt{n}$ is not the sender, then we have

$$\mathcal{G}\upharpoonright_{\mathtt{n}} = (\mathcal{G}_1\upharpoonright_{\mathtt{n}} \uplus \mathcal{G}_2\upharpoonright_{\mathtt{n}}) \uplus \mathcal{G}_3\upharpoonright_{\mathtt{n}} \quad \text{and} \quad \mathcal{G}\upharpoonright_{\mathtt{n}} = \mathcal{G}_1\upharpoonright_{\mathtt{n}} \uplus (\mathcal{G}_2\upharpoonright_{\mathtt{n}} \uplus \mathcal{G}_3\upharpoonright_{\mathtt{n}}) \tag{3.1}$$

  By Definition 3.2, either all the projections must be an external choice, or all the projections must be an internal choice (otherwise merge would not be defined).

  - If the projections are external choices, then by (3.1) and Definition 3.2, it must be the case that either all the prefixes are identical or all the prefixes are different. By contradiction, if we have

$$\mathcal{G}_1\upharpoonright_{\mathtt{n}} = \mathtt{s}?\mathsf{a} \qquad \mathcal{G}_2\upharpoonright_{\mathtt{n}} = \mathtt{s}?\mathsf{a} \qquad \mathcal{G}_3\upharpoonright_{\mathtt{n}} = \mathtt{s}?\mathsf{b}$$

    then $\mathcal{G}_1\upharpoonright_{\mathtt{n}} \uplus (\mathcal{G}_2\upharpoonright_{\mathtt{n}} \uplus \mathcal{G}_3\upharpoonright_{\mathtt{n}})$ is not defined. Similarly, if we have

$$\mathcal{G}_1\upharpoonright_{\mathtt{n}} = \mathtt{s}?\mathsf{b} \qquad \mathcal{G}_2\upharpoonright_{\mathtt{n}} = \mathtt{s}?\mathsf{a} \qquad \mathcal{G}_3\upharpoonright_{\mathtt{n}} = \mathtt{s}?\mathsf{a}$$

    then $(\mathcal{G}_1\upharpoonright_{\mathtt{n}} \uplus \mathcal{G}_2\upharpoonright_{\mathtt{n}}) \uplus \mathcal{G}_3\upharpoonright_{\mathtt{n}}$ is not defined. Thus, associating the branches differently does not affect the projection.

  - If the projections are internal choices then by (3.1) and Definition 3.2, it must be the case that all the prefixes are identical (as above). Thus, associating the branches differently does not affect the projection.

We show item 2 by induction on the structure of $\mathcal{G}$.

- If $\mathcal{G} \equiv \mu\chi.\mathcal{G}'$ the result follows by induction hypothesis and the fact $\mathcal{G}\upharpoonright_{\mathtt{n}} = \mathbf{0}$ if $\mathcal{G}'\upharpoonright_{\mathtt{n}} = \chi$.

- If $\mathcal{G} \equiv \mathbf{0}$ or $\mathcal{G} \equiv \chi$ the result holds trivially.

- If $\mathcal{G} \equiv \mathtt{s} \to \mathtt{r} : \mathtt{a}; \mathcal{G}'$ the result follows directly by induction hypothesis and Definition 3.1.

- If $\mathcal{G} \equiv \mathcal{G}_0 + \mathcal{G}_1$ we have to distinguish two cases. Either (*i*) $\mathtt{n}$ is the sender on each branch, and therefore its projections are guarded by different prefixes (cf. syntax of global types and well-formedness rules); or (*ii*) $\mathtt{n}$ is not the sender.

  In this case, we know by induction hypothesis that $\mathcal{G}_0 \upharpoonright_{\mathtt{n}}$ and $\mathcal{G}_1 \upharpoonright_{\mathtt{n}}$ are processes, we have to show that the result of $\mathcal{G}_0 \upharpoonright_{\mathtt{n}} \uplus \mathcal{G}_1 \upharpoonright_{\mathtt{n}}$ (Definition 3.2) is indeed a process. Note that since $\mathcal{G}$ is projectable, $\_ \uplus \_$ must always be defined. We show that if P and Q are processes and $P \uplus Q \neq \bot$, then $P \uplus Q$ is a process. We proceed by induction on the structure of P and Q (abstracting from commutative invocations).

  - If

    $$P \equiv \sum_{i \in I} \mathtt{s}?\mathtt{a}_\mathtt{i}; P'_i \quad \text{and} \quad Q = \sum_{j \in J} \mathtt{s}?\mathtt{a}_\mathtt{j}; Q'_j$$

    then $P \uplus Q$ is defined if

    $$\forall i \in I : \ \forall j \in J : \ \mathtt{a}_i \neq \mathtt{a}_j \text{ and } I, J \neq \varnothing$$

    and thus we have a process since both P and Q are processes and they do not share any prefix, or if $I = J$ (i.e., the prefixes are the same in both processes), and we have the result since each $P'_i \uplus Q'_j$ is a process by induction hypothesis.

  - If

    $$P \equiv \bigoplus_{i \in I} \mathtt{r}_\mathtt{i}!\mathtt{a}_\mathtt{i}; P'_i \quad \text{and} \quad Q \equiv \bigoplus_{j \in J} \mathtt{r}_\mathtt{j}!\mathtt{a}_\mathtt{j}; Q'_j$$

    then $P \uplus Q$ is defined if either $P \equiv Q$, in which case $P \uplus Q = P$, and the result follows directly; or if $I = J$ (i.e., the prefixes are the same in both processes), and we have the result since each $P'_i \uplus Q'_j$ is a process by induction hypothesis.

– If

$$P \equiv \sum_{i \in I} \mathtt{s}?\mathtt{a}_i\,;P'_i \quad \text{and} \quad Q \equiv \bigoplus_{j \in J} \mathtt{r}_j\,!\mathtt{a}_j\,;Q'_j$$

then $P \uplus Q = \bot$, i.e., a contradiction with the fact that $\mathcal{G}$ is projectable.

– If

$$P \equiv \mu\chi.P' \quad \text{or} \quad P \equiv \mathbf{0} \quad \text{or} \quad P \equiv \chi$$

then $P \uplus Q = P$ only if $P \equiv Q$, and we are done by assumption that P is a process.

- If $\mathcal{G} \equiv \mathcal{G}_0 \mid \mathcal{G}_1$, the result follows directly from induction hypothesis.

Item 3 follows directly from items 1 and 2. Item 4 follows directly from item 3 and the definition of $\mathrm{Q}(S)$. □

We state some results that were established in [40] and that are applicable in our setting for the following reasons. (*i*) Our local types can be easily translated to communicating machines, as shown in [40]. (*ii*) Our global types are essentially a subset of the *generalised global types* of [40]. Indeed, we allow only tree-shaped global types and do not allow participants to appear in different concurrent branches. (*iii*) Participants are single-threaded in our setting, thus the *linearity* condition of [40] holds trivially. (*iv*) The *local choice* condition is satisfied for all well-formed and projectable global types. This condition is satisfied for a global type $\mathcal{G}$ if for each sub-term $\mathcal{G}_0 + \mathcal{G}_1$ of $\mathcal{G}$, and for each participant $\mathtt{n} \in \mathcal{P}(\mathcal{G})$, $\mathtt{n}$ has enough information to "know" which branch of the choreography was chosen. Indeed, by Definition 3.2, if a participant has different behaviours in different branches of a global type, then these behaviours must be prefixed by different inputs.

Definition 3.3 (Properties of runtime systems [40])**.** *Let S be a runtime system, we say that*

- *S is stable if*

$$\forall \mathtt{sr} \in \mathcal{C}(S) : \; S[\mathtt{sr}] = \varepsilon$$

- *S is a deadlock if* $\forall \mathtt{sr} \in \mathcal{C}(S) : \; S[\mathtt{sr}] = \varepsilon$*, and*

$$\exists \mathtt{n} \in \mathcal{P}(S) : \; S(\mathtt{n}) \equiv \mathtt{s}?\mathtt{a}\,;P + P' \quad \textit{and} \quad \forall \mathtt{n} \in \mathcal{P}(S) : \; S(\mathtt{n}) \equiv \mathtt{s}?\mathtt{a}\,;P + P' \;\vee\; S(\mathtt{n}) \equiv \mathbf{0}$$

- *S is an orphan message configuration if*

$$\forall \mathtt{n} \in \mathcal{P}(S) : \; S(\mathtt{n}) \equiv \mathbf{0} \quad \textit{and} \quad \exists \mathtt{sr} \in \mathcal{C}(S) : \; S[\mathtt{sr}] \neq \varepsilon$$

- *S is an unspecified reception configuration if* $\exists \mathtt{r} \in \mathcal{P}(S)$:

$$S(\mathtt{r}) \equiv \mathtt{s}?\mathtt{a}\,;P + P' \implies S[\mathtt{sr}] \neq \varepsilon \;\wedge\; S[\mathtt{sr}] \neq \mathtt{a} \cdot \rho$$

A stable system is a system in which all the queues are empty. A system is a deadlock if all its queues are empty, all participants are either terminated or waiting for an input, and there is one participant expecting a message. An orphan message configuration is a system where all the participants have terminated their behaviours and there is one queue that is not empty. An unspecified reception configuration is a system for which there is a participant who is permanently unable to read a datum from one of its queues.

We can now recall the result established in [40], adapted to our setting.

Lemma 3.2 (Properties of projections [40])**.** *For all $\mathcal{G}$ such that $\vdash \mathcal{G}$ and $\mathcal{G}$ is projectable. Let*

$$S \equiv \big|_{\mathtt{n} \in \mathcal{P}(\mathcal{G})} \mathtt{n}[\mathcal{G}\!\upharpoonright_{\mathtt{n}}] \quad \textit{and} \quad \hat{S} \equiv S \mid \mathrm{Q}(S)$$

For all S' such that $\hat{S} \Longrightarrow S'$,

- *S' is not a deadlock.*

- *S' is not an orphan message configuration.*
- *S' is not an unspecified reception configuration.*
- *Either there is S'' such that $S' \to S''$, or*

$$\forall\, \mathtt{sr} \in \mathcal{C}(S) :\ S[\mathtt{sr}] = \varepsilon \quad \textit{and} \quad \forall\, \mathtt{n} \in \mathcal{P}(S) :\ S(\mathtt{n}) \equiv \mathbf{0}$$

Definition 3.4 (1-buffer execution)**.** *We say that S' is reachable from S by a* 1-buffer execution *if there exists $\{S_i \mid 1 \leqslant i \leqslant n\}$ such that $S_1 = S$, $S_n = S'$, and*

$$\forall\, 1 \leqslant i < n :\ S_i \to S_{i+1},\ \textit{and}$$
$$\exists\, \mathtt{sr} \in \mathcal{C}(S_i) :\ S_i[\mathtt{sr}] \neq \varepsilon \implies \forall\, \mathtt{s'r'} \in \mathcal{C}(S_i) \backslash \{\mathtt{sr}\} :\ S_i[\mathtt{s'r'}] = \varepsilon$$

There is a 1-buffer execution from a system S to S', if it is possible to reach S' by a series of intermediary configurations such that there is at most one non-empty buffer for each of them.

Lemma 3.3 (1-buffer execution [40])**.** *For all $\mathcal{G}$ such that $\vdash \mathcal{G}$ and $\mathcal{G}$ is projectable. Let*

$$S \equiv \big|_{\mathtt{n} \in \mathcal{P}(\mathcal{G})}\, \mathtt{n}[\mathcal{G}\upharpoonright_{\mathtt{n}}] \quad \textit{and} \quad \hat{S} \equiv S \mid \mathrm{Q}(S)$$

For all S' such that $\hat{S} \Longrightarrow S'$, if S' is stable*, then there is a 1-buffer execution from $\hat{S}$ to S'.*

3.4 Synthesising Global Types

We now introduce a type system to synthesise a global type $\mathcal{G}$ from a system S, so that S satisfies the same properties as the system consisting of the projections of a well-formed and projectable global type (cf. Lemma 3.2). Also, the system obtained by projecting a well-formed and projectable global type is always typable. One objective of the typing

system is to allow the diagram below to commute:

$$\begin{array}{ccc} S & \xrightarrow{\quad\blacktriangleright\quad} & \mathcal{G} \\ \approx \big| & & \big| \equiv \\ S' & \xleftarrow{\quad\downharpoonright\quad} & \mathcal{G}' \end{array}$$

Essentially, if a global type $\mathcal{G}$ may be synthesised ($\blacktriangleright$) from an initial system S, then $\mathcal{G}$ may be projected ($\downharpoonright$) so to obtain S' bisimilar to the original system. Analogously, the synthesis of the projections of a well-formed and projectable global type $\mathcal{G}'$ yields a structurally congruent global type $\mathcal{G}$.

To synthesise $\mathcal{G}$ from a system S, a careful analysis of what actions can occur at each possible state of S is necessary. We define the *ready set* of a system as follows:

$$\mathtt{R}(S) = \begin{cases} \{\mathtt{sr}\} \cup \mathtt{R}(S'), & \text{if } S \equiv \mathtt{r}[\sum_{i\in I} \mathtt{s}?\mathtt{a}_i\,; P_i] \mid S' \\ \{\overline{\mathtt{sr}}_i \mid i \in I\} \cup \mathtt{R}(S'), & \text{if } S \equiv \mathtt{s}[\bigoplus_{i\in I} \mathtt{r}_i!\mathtt{a}_i\,; P_i] \mid S' \\ \varnothing, & \text{if } S \equiv \mathbf{0} \end{cases}$$

We define

$$S{\updownarrow} \iff \exists\, \mathtt{sr} \in \mathbb{C} : \mathtt{sr} \in \mathtt{R}(S) \wedge \overline{\mathtt{sr}} \in \mathtt{R}(S)$$

and write $S{\not\updownarrow}$ if $S{\updownarrow}$ does not hold.

3.4.1 Validation Rules

We use judgements of the form:

$$\Gamma \vdash S \blacktriangleright \mathcal{G}$$

saying that the system S forms a choreography defined by a global type $\mathcal{G}$, under the environment Γ. The environment Γ is a map from participant names and local recursion

variables to global recursion variables. Formally, we define Γ as follows

$$\Gamma ::= \circ \mid (\mathtt{n}, \chi) : \chi' \cdot \Gamma'$$

Environment $\circ$ is the empty context, and the environment $(\mathtt{n}, \chi) : \chi'$ states that the *local* recursion variable χ of participant $\mathtt{n}$ is associated to the *global* recursion variable χ'. We write $\cdot$ for the disjoint union of environments, i.e., we write $\Gamma \cdot \Gamma'$ if $\mathtt{dom}(\Gamma) \cap \mathtt{dom}(\Gamma') = \varnothing$. The domain and image of Γ are defined below.

$$\mathtt{dom}(\Gamma) = \begin{cases} \{(\mathtt{n}, \chi)\} \cup \mathtt{dom}(\Gamma'), & \text{if } \Gamma = (\mathtt{n}, \chi) : \chi' \cdot \Gamma' \\ \varnothing, & \text{if } \Gamma = \circ \end{cases}$$

$$\mathtt{img}(\Gamma) = \begin{cases} \{\chi'\} \cup \mathtt{img}(\Gamma'), & \text{if } \Gamma = (\mathtt{n}, \chi) : \chi' \cdot \Gamma' \\ \varnothing, & \text{if } \Gamma = \circ \end{cases}$$

A global type $\mathcal{G}$ can be synthesised from the *program* S if the judgement

$$\circ \vdash S \triangleright \mathcal{G}$$

is derivable from the rules in Figure 3.3. The rules are driven by the ready set of S and the structure of its processes, we detail each of them below.

Sequential interactions are dealt with by rule [;]. The rule validates prefixes provided that the continuation is typable and that no other interactions are possible in S. For instance, rule [;] does not apply to

$$\mathtt{s}_1[\mathtt{r}_1!\mathtt{a}; P_1] \mid \mathtt{r}_1[\mathtt{s}_1?\mathtt{a}; Q_1] \mid \mathtt{s}_2[\mathtt{r}_2!\mathtt{b}; P_2] \mid \mathtt{r}_2[\mathtt{s}_2?\mathtt{b}; Q_2] \qquad \text{✘}$$

because there is no ordering relation between the actions of $\mathtt{s}_1$ and $\mathtt{r}_1$ on one hand, and the actions of $\mathtt{s}_2$ and $\mathtt{r}_2$ on the other hand. In such cases, the rule [|] should be used.

$$[;]\ \frac{\Gamma \vdash \mathtt{s}[P] \mid \mathtt{r}[Q] \mid S \triangleright \mathcal{G} \qquad S\not\updownarrow}{\Gamma \vdash \mathtt{s}[\mathtt{r}!\mathtt{a};P] \mid \mathtt{r}[\mathtt{s}?\mathtt{a};Q] \mid S \triangleright \mathtt{s} \to \mathtt{r}:\mathtt{a};\mathcal{G}}$$

$$[\,|\,]\ \frac{\circ \vdash S \triangleright \mathcal{G} \qquad \circ \vdash S' \triangleright \mathcal{G}'}{\Gamma \vdash S \mid S' \triangleright \mathcal{G} \mid \mathcal{G}'}$$

$$[\oplus]\ \frac{\Gamma \vdash \mathtt{s}[P] \mid S \triangleright \mathcal{G} \qquad \Gamma \vdash \mathtt{s}[Q] \mid S \triangleright \mathcal{G}' \qquad S\not\updownarrow}{\Gamma \vdash \mathtt{s}[P \oplus Q] \mid S \triangleright \mathcal{G} + \mathcal{G}'}$$

$$[+]\ \frac{\Gamma \vdash \mathtt{r}[P] \mid S \triangleright \mathcal{G}}{\Gamma \vdash \mathtt{r}[P + Q] \mid S \triangleright \mathcal{G}}$$

$$[\mu]\ \frac{\begin{array}{c}\exists\, 1 \leqslant \mathtt{i},\mathtt{j} \leqslant \mathtt{k} :\ (\mathtt{n}_\mathtt{i}[P_\mathtt{i}] \mid \mathtt{n}_\mathtt{j}[P_\mathtt{j}])\updownarrow \\ \Gamma \cdot (\mathtt{n}_1,\chi_1) : \chi, \ldots, (\mathtt{n}_\mathtt{k},\chi_\mathtt{k}) : \chi \vdash \mathtt{n}_1[P_1] \mid \ldots \mid \mathtt{n}_\mathtt{k}[P_\mathtt{k}] \triangleright \mathcal{G}\end{array}}{\Gamma \vdash \mathtt{n}_1[\mu\chi_1.P_1] \mid \ldots \mid \mathtt{n}_\mathtt{k}[\mu\chi_\mathtt{k}.P_\mathtt{k}] \triangleright \mu\chi.\mathcal{G}}$$

$$[\chi]\ \frac{\forall 1 \leqslant \mathtt{i} \leqslant k :\ \Gamma(\mathtt{n}_\mathtt{i},\chi_\mathtt{i}) = \chi}{\Gamma \vdash \mathtt{n}_1[\chi_1] \mid \ldots \mid \mathtt{n}_\mathtt{k}[\chi_\mathtt{k}] \triangleright \chi}$$

$$[\equiv]\ \frac{S \equiv S' \qquad \Gamma \vdash S' \triangleright \mathcal{G}}{\Gamma \vdash S \triangleright \mathcal{G}} \qquad [\mathbf{0}]\ \frac{\forall \mathtt{n} \in \mathcal{P}(S) :\ S(\mathtt{n}) = \mathbf{0}}{\Gamma \vdash S \triangleright \mathbf{0}}$$

Figure 3.3: Validation rules for programs

Concurrent branches are introduced by rule $[\,|\,]$. The rule validates concurrent branches when they can be validated using a partition of the system begin considered, recall that $\mathcal{P}(S) \cap \mathcal{P}(S') = \varnothing$.

Choice in a global type is dealt with by rules $[\oplus]$ and $[+]$. Rule $[\oplus]$ introduces the global type choice operator, it requires that both branches are typable and that no other interactions are possible in S – for the same reason as in rule $[;]$. Rule $[+]$ allows to discharge a branch of an external choice. This allows to type systems of the form:

$$\mathtt{s}[\mathtt{r}!\mathtt{a} \oplus \mathtt{r}!\mathtt{b}] \mid \mathtt{r}[\mathtt{s}?\mathtt{a} + \mathtt{s}?\mathtt{b} + \mathtt{s}?\mathtt{c}] \qquad \checkmark$$

Recursion is handled by rules $[\mu]$ and $[\chi]$. The former rule "guesses" the participants involved in a recursive behaviour. If two of them interact, $[\mu]$ validates the recursion provided that the system can be typed when such participants are associated to the global recursion variable χ – assuming that $\chi \notin \mathtt{img}(\Gamma)$. Rule $[\chi]$ checks that all the participants in the recursion have reached a local recursion variable corresponding to the global recursion.

The termination $\mathbf{0}$ is introduced by rule $[\mathbf{0}]$, which only applies when all the participants in S are terminated.

Rule $[\equiv]$ validates a system up to structural congruence. This rule notably allows recursive behaviours to be unfolded, see Example 3.7 below.

3.4.2 Applying the Rules

We give a few examples of derivations. We give the full derivation of the system S_{BS} from Section 3.1, show that we support an example borrowed from [41], and we illustrate the use of rule $[\equiv]$ to unfold behaviours.

First, we show how to synthesise the global type corresponding to system S_{BS} of Section 3.1. Recall that this system is defined as follows:

$$S_{\mathrm{BS}} \;=\; \mathtt{b}_1[P_1] \;\mid\; \mathtt{s}_1[S_1] \;\mid\; \mathtt{b}_2[P_2] \;\mid\; \mathtt{s}_2[S_2]$$

with

$$\begin{array}{lcll} P_1 & = & \mathtt{b}_2!\mathsf{agreement};\mathtt{b}_2?\mathsf{ack};Q_1 & \\ P_2 & = & \mathtt{b}_1?\mathsf{agreement};\mathtt{b}_1!\mathsf{ack};Q_2 & \\ Q_{\mathtt{i}} & = & \mathtt{s}_{\mathtt{i}}!\mathsf{request};\mathtt{s}_{\mathtt{i}}?\mathsf{quote};\mu\chi.(\mathtt{s}_{\mathtt{i}}!\mathsf{ok} \oplus \mathtt{s}_{\mathtt{i}}!\mathsf{no};\mathtt{s}_{\mathtt{i}}?\mathsf{quote};\chi) & \mathtt{i} \in \{1,2\} \\ S_{\mathtt{i}} & = & \mathtt{b}_{\mathtt{i}}?\mathsf{request};\mathtt{b}_{\mathtt{i}}!\mathsf{quote};\mu\chi'.(\mathtt{b}_{\mathtt{i}}?\mathsf{ok} + \mathtt{b}_{\mathtt{i}}?\mathsf{no};\mathtt{b}_{\mathtt{i}}!\mathsf{quote};\chi') & \mathtt{i} \in \{1,2\} \end{array}$$

We describe the derivation from the rule typing the overall system to the leaves of the type derivation. The derivation of the overall system S_{BS} is as follows.

$$
\dfrac{\dfrac{\dfrac{\nabla_l \qquad \nabla_r}{\circ \vdash \mathtt{b}_1[Q_1] \mid \mathtt{s}_1[S_1] \triangleright \mathcal{G}_2^1 \qquad \circ \vdash \mathtt{b}_2[Q_2] \mid \mathtt{s}_2[S_2] \triangleright \mathcal{G}_2^2}\,[\,|\,]}{\circ \vdash \mathtt{b}_1[Q_1] \mid \mathtt{s}_1[S_1] \mid \mathtt{b}_2[Q_2] \mid \mathtt{s}_2[S_2] \triangleright \mathcal{G}_2^1 \mid \mathcal{G}_2^2}\,[;]}{\circ \vdash \mathtt{b}_1[\mathtt{b}_2?\mathsf{ack};Q_1] \mid \mathtt{s}_1[S_1] \mid \mathtt{b}_2[\mathtt{b}_1!\mathsf{ack};Q_2] \mid \mathtt{s}_2[S_2] \triangleright \mathtt{b}_2 \to \mathtt{b}_1 : \mathsf{ack};\, (\mathcal{G}_2^1 \mid \mathcal{G}_2^2)}\,[;]}{\circ \vdash \mathtt{b}_1[P_1] \mid \mathtt{s}_1[S_1] \mid \mathtt{b}_2[P_2] \mid \mathtt{s}_2[S_2] \triangleright \mathcal{G}_{S_{\text{BS}}}}
$$

First, rule $[;]$ is applied twice, recording the two interactions between $\mathtt{b}_1$ and $\mathtt{b}_2$. Then, rule $[\,|\,]$ is applied to separate the system in two concurrent sub-systems. The derivation of the left (∇_l) and right (∇_r) branches are similar and we give a parametrised version of them below. We pose

$$
\begin{aligned}
\hat{\mathcal{G}}_r &= \mathtt{b}_\mathtt{i} \to \mathtt{s}_\mathtt{i} : \mathsf{ok} + \mathtt{b}_\mathtt{i} \to \mathtt{s}_\mathtt{i} : \mathsf{no};\, \mathtt{s}_\mathtt{i} \to \mathtt{b}_\mathtt{i} : \mathsf{quote};\, \chi \\
\hat{Q}_r &= \mathtt{s}_\mathtt{i}!\mathsf{ok} \oplus \mathtt{s}_\mathtt{i}!\mathsf{no};\mathtt{s}_\mathtt{i}?\mathsf{quote};\chi \\
\hat{S}_r &= \mathtt{b}_\mathtt{i}?\mathsf{ok} + \mathtt{b}_\mathtt{i}?\mathsf{no};\mathtt{b}_\mathtt{i}!\mathsf{quote};\chi'
\end{aligned}
$$

in the following derivation:

$$
\dfrac{\dfrac{\dfrac{\dfrac{\nabla_{\text{in-rec}}}{(\mathtt{b}_\mathtt{i},\chi) : \chi \cdot (\mathtt{s}_\mathtt{i},\chi') : \chi \vdash \mathtt{b}_\mathtt{i}\big[\hat{Q}_r\big] \mid \mathtt{s}_\mathtt{i}\big[\hat{S}_r\big] \triangleright \hat{\mathcal{G}}_r}\,[\mu]}{\circ \vdash \mathtt{b}_\mathtt{i}\big[\mu\chi.\hat{Q}_r\big] \mid \mathtt{s}_\mathtt{i}\big[\mu\chi'.\hat{S}_r\big] \triangleright \mu\chi.\hat{\mathcal{G}}_r}\,[;]}{\circ \vdash \mathtt{b}_\mathtt{i}\big[\mathtt{s}_\mathtt{i}?\mathsf{quote};\mu\chi.\hat{Q}_r\big] \mid \mathtt{s}_\mathtt{i}\big[\mathtt{b}_\mathtt{i}!\mathsf{quote};\mu\chi'.\hat{S}_r\big] \triangleright \mathtt{s}_\mathtt{i} \to \mathtt{b}_\mathtt{i} : \mathsf{quote};\, \mu\chi.\hat{\mathcal{G}}_r}\,[;]}{\circ \vdash \mathtt{b}_\mathtt{i}[Q_\mathtt{i}] \mid \mathtt{s}_\mathtt{i}[S_\mathtt{i}] \triangleright \mathcal{G}_2^\mathtt{i}}
$$

In the sub-derivation above, rule $[;]$ is applied twice again to record the interactions be-

tween $\mathtt{b_i}$ and $\mathtt{s_i}$, then rule $[\mu]$ is applied and the recursion environment is updated correspondingly. The next step in the sub-derivation $\nabla_{\text{in-rec}}$ is given below:

$$[\oplus]\ \frac{\nabla_{\text{end}} \qquad \nabla_{\text{rec}}}{(\mathtt{b_i},\chi):\chi\cdot(\mathtt{s_i},\chi'):\chi \vdash \mathtt{b_i}\left[\hat{Q}_r\right]\ \mid \mathtt{s_i}\left[\hat{S}_r\right] \triangleright \hat{\mathcal{G}}_r}$$

where $[\oplus]$ is used to split the derivation in two parts, one per branch of the internal choice in $\mathtt{b_i}$, both sub-derivations are given below. First, we give ∇_{end}:

$$\cfrac{\cfrac{\cfrac{}{(\mathtt{b_i},\chi):\chi\cdot(\mathtt{s_i},\chi'):\chi \vdash \mathtt{b_i}[\mathbf{0}]\ \mid \mathtt{s_i}[\mathbf{0}] \triangleright \mathbf{0}}[\mathbf{0}]}{(\mathtt{b_i},\chi):\chi\cdot(\mathtt{s_i},\chi'):\chi \vdash \mathtt{b_i}[\mathtt{s_i}!\mathsf{ok}]\ \mid \mathtt{s_i}[\mathtt{b_i}?\mathsf{ok}] \triangleright \mathtt{b_i}\to\mathtt{s_i}:\mathsf{ok}}[;]}{(\mathtt{b_i},\chi):\chi\cdot(\mathtt{s_i},\chi'):\chi \vdash \mathtt{b_i}[\mathtt{s_i}!\mathsf{ok}]\ \mid \mathtt{s_i}\left[\hat{S}_r\right] \triangleright \mathtt{b_i}\to\mathtt{s_i}:\mathsf{ok}}[+]$$

In the sub-derivation above, rule $[+]$ is used to discard the right branch of $\mathtt{s_i}$, then rule $[;]$ is used for the ok interaction. The last rule applied is rule $[\mathbf{0}]$ which terminates the global type.

We now give ∇_{rec}, where we pose $\hat{\mathcal{G}}_l = \mathtt{b_i}\to\mathtt{s_i}:\mathsf{no};\ \mathtt{s_i}\to\mathtt{b_i}:\mathsf{quote};\ \chi$.

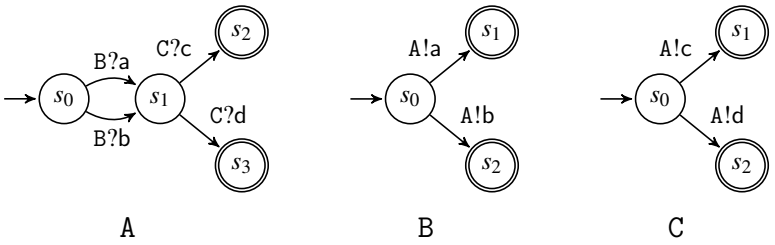
$$\cfrac{\cfrac{\cfrac{\cfrac{}{(\mathtt{b_i},\chi):\chi\cdot(\mathtt{s_i},\chi'):\chi \vdash \mathtt{b_i}[\chi]\ \mid \mathtt{s_i}\left[\chi'\right] \triangleright \chi}[\chi]}{(\mathtt{b_i},\chi):\chi\cdot(\mathtt{s_i},\chi'):\chi \vdash \mathtt{b_i}[\mathtt{s_i}?\mathsf{quote};\chi]\ \mid \mathtt{s_i}\left[\mathtt{b_i}!\mathsf{quote};\chi'\right] \triangleright \mathtt{s_i}\to\mathtt{b_i}:\mathsf{quote};\ \chi}[;]}{(\mathtt{b_i},\chi):\chi\cdot(\mathtt{s_i},\chi'):\chi \vdash \mathtt{b_i}[\mathtt{s_i}!\mathsf{no};\mathtt{s_i}?\mathsf{quote};\chi]\ \mid \mathtt{s_i}\left[\mathtt{b_i}?\mathsf{no};\mathtt{b_i}!\mathsf{quote};\chi'\right] \triangleright \hat{\mathcal{G}}_l}[;]}{(\mathtt{b_i},\chi):\chi\cdot(\mathtt{s_i},\chi'):\chi \vdash \mathtt{b_i}[\mathtt{s_i}!\mathsf{no};\mathtt{s_i}?\mathsf{quote};\chi]\ \mid \mathtt{s_i}\left[\hat{S}_r\right] \triangleright \hat{\mathcal{G}}_l}[+]$$

In this, case we use rule $[+]$ to discard the left branch of $\mathtt{s_i}$, then use rule $[;]$ twice, and finally we apply rule $[\chi]$ to introduce the recursion variable. Note that this rule is applicable

since the recursion environment maps each local recursion variable to the same global recursion variable.

We now give two additional examples of typing derivations. Example 3.6 shows the typing of local types obtained from communicating machines given in Remark 4.1 of [41]. Example 3.7 illustrates the use of rule $[\equiv]$, whose main role is to unfold behaviours when required.

Example 3.6. Consider the following communicating machines (borrowed from [41]):



These machines may be easily translated to our local types so to obtain the following system:

$$S_{\text{ex3.6}} \;=\; \mathtt{A}[\mathtt{B?a};(\mathtt{C?c}+\mathtt{C?d})+\mathtt{B?b};(\mathtt{C?c}+\mathtt{C?d})] \;\mid\; \mathtt{B}[\mathtt{A!a}\oplus\mathtt{A!b}] \;\mid\; \mathtt{C}[\mathtt{A!c}\oplus\mathtt{A!d}]$$

In fact, we have $\circ \vdash S_{\text{ex3.6}} \triangleright \mathcal{G}_{\text{ex3.6}}$, with $\mathcal{G}_{\text{ex3.6}}$ defined as follows:

$$\mathcal{G}_{\text{ex3.6}} \;=\; \mathtt{B}\to\mathtt{A}:\mathtt{a};\,\mathcal{G}' \;+\; \mathtt{B}\to\mathtt{A}:\mathtt{b};\,\mathcal{G}' \qquad \mathcal{G}' \;=\; \mathtt{C}\to\mathtt{A}:\mathtt{c} \;+\; \mathtt{C}\to\mathtt{A}:\mathtt{d}$$

The complete typing derivation is given in Figure 3.4. ◇

Example 3.7. Consider the system below.

$$S_{\text{ex3.7}} = \mathtt{s}[\mathtt{r!a};\mathtt{r?b};\mu\chi.\mathtt{r!a};\mathtt{r?b};\chi] \;\mid\; \mathtt{r}[\mu\chi.\mathtt{s?a};\mathtt{s!b};\chi]$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\nabla}{\circ \vdash \mathtt{A}[\mathtt{C}?\mathtt{c}+\mathtt{C}?\mathtt{d}] \mid \mathtt{B}[\mathbf{0}] \mid \mathtt{C}[\ldots] \triangleright \mathcal{G}'}[;]}{\circ \vdash \mathtt{A}[\mathtt{B}?\mathtt{a};(\ldots)] \mid \mathtt{B}[\mathtt{A}!\mathtt{a}] \mid \mathtt{C}[\ldots] \triangleright \mathtt{B}\to\mathtt{A}:\mathtt{a};\mathcal{G}'}[+] \quad \dfrac{\dfrac{\nabla}{\circ \vdash \mathtt{A}[\mathtt{C}?\mathtt{c}+\mathtt{C}?\mathtt{d}] \mid \mathtt{B}[\mathbf{0}] \mid \mathtt{C}[\ldots] \triangleright \mathcal{G}'}[;]}{\circ \vdash \mathtt{A}[\mathtt{B}?\mathtt{b};(\ldots)] \mid \mathtt{B}[\mathtt{A}!\mathtt{b}] \mid \mathtt{C}[\ldots] \triangleright \mathtt{B}\to\mathtt{A}:\mathtt{b};\mathcal{G}'}[+]}{\circ \vdash \mathtt{A}[\ldots] \mid \mathtt{B}[\mathtt{A}!\mathtt{a}] \mid \mathtt{C}[\ldots] \triangleright \mathtt{B}\to\mathtt{A}:\mathtt{a};\mathcal{G}'}[\oplus]}{\circ \vdash \mathtt{A}[\mathtt{B}?\mathtt{a};(\mathtt{C}?\mathtt{c}+\mathtt{C}?\mathtt{d})+\mathtt{B}?\mathtt{b};(\mathtt{C}?\mathtt{c}+\mathtt{C}?\mathtt{d})] \mid \mathtt{B}[\mathtt{A}!\mathtt{a}\oplus\mathtt{A}!\mathtt{b}] \mid \mathtt{C}[\mathtt{A}!\mathtt{c}\oplus\mathtt{A}!\mathtt{d}] \triangleright \mathcal{G}}$$

$$\nabla \;=\; \dfrac{\dfrac{\dfrac{\dfrac{[\mathbf{0}]\ \circ \vdash \mathtt{A}[\mathbf{0}] \mid \mathtt{B}[\mathbf{0}] \mid \mathtt{C}[\mathbf{0}] \triangleright \mathbf{0}}{\circ \vdash \mathtt{A}[\mathtt{C}?\mathtt{c}] \mid \mathtt{B}[\mathbf{0}] \mid \mathtt{C}[\mathtt{A}!\mathtt{c}] \triangleright \mathtt{C}\to\mathtt{A}:\mathtt{c}}[;]}{\circ \vdash \mathtt{A}[\mathtt{C}?\mathtt{c}+\mathtt{C}?\mathtt{d}] \mid \mathtt{B}[\mathbf{0}] \mid \mathtt{C}[\mathtt{A}!\mathtt{c}] \triangleright \mathtt{C}\to\mathtt{A}:\mathtt{c}}[+] \quad \dfrac{\dfrac{[\mathbf{0}]\ \circ \vdash \mathtt{A}[\mathbf{0}] \mid \mathtt{B}[\mathbf{0}] \mid \mathtt{C}[\mathbf{0}] \triangleright \mathbf{0}}{\circ \vdash \mathtt{A}[\mathtt{C}?\mathtt{d}] \mid \mathtt{B}[\mathbf{0}] \mid \mathtt{C}[\mathtt{A}!\mathtt{d}] \triangleright \mathtt{C}\to\mathtt{A}:\mathtt{d}}[;]}{\circ \vdash \mathtt{A}[\mathtt{C}?\mathtt{c}+\mathtt{C}?\mathtt{d}] \mid \mathtt{B}[\mathbf{0}] \mid \mathtt{C}[\mathtt{A}!\mathtt{d}] \triangleright \mathtt{C}\to\mathtt{A}:\mathtt{d}}[+]}{\circ \vdash \mathtt{A}[\mathtt{C}?\mathtt{c}+\mathtt{C}?\mathtt{d}] \mid \mathtt{B}[\mathbf{0}] \mid \mathtt{C}[\mathtt{A}!\mathtt{c}\oplus\mathtt{A}!\mathtt{d}] \triangleright \mathcal{G}'}[\oplus]$$

Figure 3.4: Typing derivation of $S_{\text{ex3.6}}$

We have the following derivation

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{(\mathtt{s},\chi):\chi\cdot(\mathtt{r},\chi):\chi \vdash \mathtt{s}[\mathtt{r}!\mathtt{a};\mathtt{r}?\mathtt{b};\chi] \mid \mathtt{r}[\mathtt{s}?\mathtt{a};\mathtt{s}!\mathtt{b};\chi] \triangleright \mathcal{G}''}[;]}{\circ \vdash \mathtt{s}[\mu\chi.\mathtt{r}!\mathtt{a};\mathtt{r}?\mathtt{b};\chi] \mid \mathtt{r}[\mu\chi.\mathtt{s}?\mathtt{a};\mathtt{s}!\mathtt{b};\chi] \triangleright \mu\chi.\mathcal{G}''}[\mu]}{\circ \vdash \mathtt{s}[\mathtt{r}?\mathtt{b};\mu\chi.\mathtt{r}!\mathtt{a};\mathtt{r}?\mathtt{b};\chi] \mid \mathtt{r}[\mathtt{s}!\mathtt{b};\mu\chi.\mathtt{s}?\mathtt{a};\mathtt{s}!\mathtt{b};\chi] \triangleright \mathcal{G}'}[;]}{\circ \vdash \mathtt{s}[\mathtt{r}!\mathtt{a};\mathtt{r}?\mathtt{b};\mu\chi.\mathtt{r}!\mathtt{a};\mathtt{r}?\mathtt{b};\chi] \mid \mathtt{r}[\mathtt{s}?\mathtt{a};\mathtt{s}!\mathtt{b};\mu\chi.\mathtt{s}?\mathtt{a};\mathtt{s}!\mathtt{b};\chi] \triangleright \mathcal{G}_{\text{ex3.7}}}[;]}{\circ \vdash S_{\text{ex3.7}} \triangleright \mathcal{G}_{\text{ex3.7}}}[\equiv]$$

Where

$$\mathcal{G}_{\text{ex3.7}} = \mathtt{s}\to\mathtt{r}:\mathtt{a};\mathcal{G}' \qquad \mathcal{G}' = \mathtt{r}\to\mathtt{s}:\mathtt{b};\mu\chi.\mathcal{G}'' \qquad \mathcal{G}'' = \mathtt{s}\to\mathtt{r}:\mathtt{a};\mathtt{r}\to\mathtt{s}:\mathtt{b};\chi$$

It is clear that the only rule applicable to type $S_{\text{ex3.7}}$ is rule [≡], since neither there are two behaviours with matching prefixes, nor all processes are recursive. ◇

3.5 Properties of the Synthesis

In this section, we show the properties of the type system, which, notably, allow us to prove that the diagram of Section 3.4 commutes. We summarise our results below.

- Theorem 3.1 gives safety and progress properties for typable systems.
- Theorem 3.2 states that typability is decidable.
- Theorem 3.3 states that the global type assigned to a system is unique (up to structural congruence).
- Lemma 3.8 characterises the relationship between the behaviours in the initial system and the projections of the synthesised global type. This lemma is crucial to show that a synthesised global type is always well-formed (Theorem 3.4).
- Theorem 3.5 gives a subject reduction result.
- Theorem 3.6 shows that the system consisting of the projections of a synthesised global type is bisimilar to the original system.
- Theorem 3.7 shows that any well-formed and projectable global type may be projected and synthesised again to an equivalent global type.

The following intermediary result follows directly from the rules of Figure 3.3.

Lemma 3.4. *If* $\Gamma \vdash S \blacktriangleright \mathcal{G}$ *then* $\forall \mathtt{n} \in \mathcal{P}(S) : S(\mathtt{n}) \neq \mathbf{0} \iff \mathtt{n} \in \mathcal{P}(\mathcal{G})$

Proof. Straightforward induction on the derivation. □

Theorem 3.1 below follows from the results in the rest of this section. Namely, we will show that a typable system and the system consisting of its projections are bisimilar, and that a synthesised global type is well-formed. This allows us to reuse the results of Lemma 3.2 to guarantee the same safety properties as with the usual top-down approach of the multiparty session types.

Theorem 3.1 (Properties of typable systems)**.** *Let S be a stable runtime system, such that $\circ \vdash S \triangleright \mathcal{G}$. For all S' such that $S \Longrightarrow S'$,*

- *S' is not a deadlock.*
- *S' is not an orphan message configuration.*
- *S' is not an unspecified reception configuration.*
- *Either there is S'' such that $S' \to S''$, or*

$$\forall\, \mathtt{sr} \in \mathcal{C}(S') :\ S'[\mathtt{sr}] = \varepsilon \quad \textit{and} \quad \forall\, \mathtt{n} \in \mathcal{P}(S') :\ S'(\mathtt{n}) \equiv \mathbf{0}$$

- *If S and S' are* stable*, then there is a 1-buffer execution from S to S'.*

Proof. The result follows directly from the fact that $\mathcal{G}$ is a well-formed global type (Theorem 3.4), the projections of a well-formed global types guarantee such properties (Lemmas 3.2 and 3.3), and the projections of a synthesised global type are $\approx$-equivalent to the original system (Theorem 3.6). □

3.5.1 Decidability

A crucial result is that it is decidable whether or not a system may be assigned a global type. We give a few examples of type derivations to show that the answer to this question may not be straightforward.

Example 3.8. Consider the typable system:

$$\mathtt{s}[\mu\chi.\mathtt{r}!\mathtt{a};\chi] \ \mid\ \mathtt{r}[\mu\chi.\mathtt{s}?\mathtt{a};\chi]$$

A possibly infinite derivation for this system would be of the form

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\circ \vdash \mathtt{s}[\mu\chi.\mathtt{r}!\mathtt{a};\chi] \mid \mathtt{r}[\mu\chi.\mathtt{s}?\mathtt{a};\chi] \triangleright \mathcal{G}}\,[\equiv]}{\circ \vdash \mathtt{s}[\mathtt{r}!\mathtt{a};\mu\chi.\mathtt{r}!\mathtt{a};\chi] \mid \mathtt{r}[\mathtt{s}?\mathtt{a};\mu\chi.\mathtt{s}?\mathtt{a};\chi] \triangleright \mathtt{s}\to\mathtt{r}:\mathtt{a};\, \mathcal{G}}\,[;]}{\circ \vdash \mathtt{s}[\mu\chi.\mathtt{r}!\mathtt{a};\chi] \mid \mathtt{r}[\mu\chi.\mathtt{s}?\mathtt{a};\chi] \triangleright \mathtt{s}\to\mathtt{r}:\mathtt{a};\, \mathcal{G}}\,[\equiv]$$

where rules $[\equiv]$ and $[;]$ are applied successively, possibly endlessly. Of course, if we apply rule $[\mu]$ first we are done in a few steps:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{(\mathtt{s},\chi):\chi\cdot(\mathtt{r},\chi):\chi \vdash \mathtt{s}[\chi] \mid \mathtt{r}[\chi] \triangleright \chi}\,[\chi]}{(\mathtt{s},\chi):\chi\cdot(\mathtt{r},\chi):\chi \vdash \mathtt{s}[\mathtt{r}!\mathtt{a};\chi] \mid \mathtt{r}[\mathtt{s}?\mathtt{a};\chi] \triangleright \mathtt{s}\to\mathtt{r}:\mathtt{a};\chi}\,[;]}{\circ \vdash \mathtt{s}[\mu\chi.\mathtt{r}!\mathtt{a};\chi] \mid \mathtt{r}[\mu\chi.\mathtt{s}?\mathtt{a};\chi] \triangleright \mu\chi.\mathtt{s}\to\mathtt{r}:\mathtt{a};\chi}\,[\mu]$$

$\diamond$

Example 3.9. Consider the system

$$\mathtt{s}[\mathtt{r}!\mathtt{b};\mu\chi.\mathtt{r}!\mathtt{b};\mathtt{r}!\mathtt{a};\chi] \mid \mathtt{r}[\mu\chi.\mathtt{s}?\mathtt{b};\mathtt{s}?\mathtt{a};\chi]$$

This system is not typable: consider the following partial derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\bot}{\circ \vdash \mathtt{s}[\mu\chi.\mathtt{r}!\mathtt{b};\mathtt{r}!\mathtt{a};\chi] \mid \mathtt{r}[\mathtt{s}?\mathtt{a};\mu\chi.\mathtt{s}?\mathtt{b};\mathtt{s}?\mathtt{a};\chi] \triangleright \ldots}\,[;]}{\circ \vdash \mathtt{s}[\mathtt{r}!\mathtt{b};\mu\chi.\mathtt{r}!\mathtt{b};\mathtt{r}!\mathtt{a};\chi] \mid \mathtt{r}[\mathtt{s}?\mathtt{b};\mathtt{s}?\mathtt{a};\mu\chi.\mathtt{s}?\mathtt{b};\mathtt{s}?\mathtt{a};\chi] \triangleright \ldots}\,[\equiv]}{\circ \vdash \mathtt{s}[\mathtt{r}!\mathtt{b};\mu\chi.\mathtt{r}!\mathtt{b};\mathtt{r}!\mathtt{a};\chi] \mid \mathtt{r}[\mu\chi.\mathtt{s}?\mathtt{b};\mathtt{s}?\mathtt{a};\chi] \triangleright \ldots}$$

Intuitively, this is due to the fact that participant $\mathtt{s}$ is ready to communicate with a participant ($\mathtt{r}$) which exhibits directly a recursive behaviour. However, the sub-term $\mathtt{r}!\mathtt{b};\mathbf{0}$ (i.e., the part of the behaviour outside the recursion, appended by $\mathbf{0}$) is not the same as behaviour within the recursion, i.e., $\mathtt{r}!\mathtt{b};\mathtt{r}!\mathtt{a};\mathbf{0}$. Thus, there is a mismatch between the behaviour of $\mathtt{s}$ outside the recursion and the one of $\mathtt{r}$ within the recursion. $\diamond$

We now introduce a few definitions that we will use in the proof of Theorem 3.2 below.

Definition 3.5 (Unfolding)**.** *Let* $\mathtt{unfold}_i(P)$ *be the i-time unfolding of P, defined as follows*

$$\mathtt{unfold}_i(P) = \mathtt{unfold}_1(\mathtt{unfold}_{i-1}(P)) \quad i > 1$$

$$\mathtt{unfold}_1(P) = \begin{cases} \bigoplus_{i\in I} \mathtt{r}_\mathtt{i}!\mathtt{a}_\mathtt{i};\mathtt{unfold}_1(P_i) & \text{if } P = \bigoplus_{i\in I} \mathtt{r}_\mathtt{i}!\mathtt{a}_\mathtt{i};P_i \\ \sum_{i\in I} \mathtt{s}?\mathtt{a}_\mathtt{i};\mathtt{unfold}_1(P_i) & \text{if } P = \sum_{i\in I} \mathtt{s}?\mathtt{a}_\mathtt{i};P_i \\ \chi & \text{if } P = \chi \\ P'\{\mu\chi.P'/\chi\} & \text{if } P = \mu\chi.P' \end{cases}$$

We illustrate this definition with the example below.

Example 3.10. Consider the following behaviours

$$P = \mu\chi.P' \qquad P' = \mathtt{r}!\mathtt{a};\mu\chi'.(\mathtt{r}!\mathtt{b};\chi \oplus \mathtt{r}!\mathtt{c};\chi')$$

The one-time unfolding of P is

$$\mathtt{unfold}_1(P) = \mathtt{r}!\mathtt{a};\mu\chi'.(\mathtt{r}!\mathtt{b};P \oplus \mathtt{r}!\mathtt{c};\chi')$$

while the one-time unfolding of P' is

$$\mathtt{unfold}_1(P') = \mathtt{r}!\mathtt{a};\left(\mathtt{r}!\mathtt{b};\chi \oplus \mathtt{r}!\mathtt{c};\mu\chi'.\left(\mathtt{r}!\mathtt{b};\chi \oplus \mathtt{r}!\mathtt{c};\chi'\right)\right)$$

◇

Definition 3.6 (Behaviour context)**.** *Let a (non-recursive) behaviour context $C\langle\_\rangle$ be defined as follows*

$$C\langle\_\rangle ::= \bigoplus_{i\in I} \mathtt{r}_i!\mathtt{a}_i;C_i\langle\_\rangle \;\mid\; \sum_{i\in I} \mathtt{s}?\mathtt{a}_i;C_i\langle\_\rangle \;\mid\; \langle\_\rangle$$

and a (possibly recursive) behaviour context $C'\langle\_\rangle$ be defined as follows

$$C'\langle\_\rangle ::= \bigoplus_{i\in I} \mathtt{r}_i!\mathtt{a}_i;C'_i\langle\_\rangle \;\mid\; \sum_{i\in I} \mathtt{s}?\mathtt{a}_i;C'_i\langle\_\rangle \;\mid\; \langle\_\rangle \;\mid\; \mu\chi.C'\langle\_\rangle \;\mid\; \chi$$

A non-recursive context $C\langle\_\rangle$ is simply a context where no recursive definition may appear, while a possibly recursive context $C'\langle\_\rangle$ may be any context defined by the syntax of behaviours. We illustrate this definition in Example 3.11 below.

Using Definitions 3.6 and 3.5, we define a "distance" between the non-recursive context of a behaviour and its unfolding.

Definition 3.7. *Given a process of the form:*

$$\mathtt{n}\big[C\big\langle\mu\chi.C'\langle\chi\rangle\big\rangle\big] \qquad \textit{let } P_k = \mathtt{unfold}_k(\mu\chi.C'\langle\chi\rangle)$$

We define $|\mathtt{n}|$ to be the smallest k such that $C\langle\mathbf{0}\rangle$ is a sub-tree of P_k where all sub-terms of the form $\mu\chi.P'$ in P_k are replaced by $\mathbf{0}$; and $|\mathtt{n}| = \bot$ if there is no such k.

Note that if $|\mathtt{n}|$ is defined, it must be smaller or equal than the length of $C\langle\mathbf{0}\rangle$ (since recursion is guarded).

Example 3.11. We illustrate Definition 3.7.

- Assume we have $\mathtt{n}$ such that $S(\mathtt{n}) = P$, and

$$P \;=\; C\big\langle\mu\chi.C'\langle\chi\rangle\big\rangle \;=\; \mathtt{r}!\mathtt{a};\mathtt{r}!\mathtt{a};\mu\chi.\mathtt{r}!\mathtt{a};\chi$$

We have $|\mathtt{n}| = 2$ since

$$\mathtt{unfold}_2(\mu\chi.\mathtt{r!a};\chi) = \mathtt{r!a};\mathtt{r!a};\mu\chi.\mathtt{r!a};\chi$$

- Assume we have $\mathtt{n}$ such that $S(\mathtt{n}) = P$, and

$$P \;=\; C\langle\mu\chi.C'\langle\chi\rangle\rangle \;=\; \mathtt{r!a};\mu\chi.(\mathtt{r!a};\chi\oplus\mathtt{r!b};\chi)$$

We have $|\mathtt{n}| = 1$ since $\mathtt{unfold}_1(\mu\chi.(\mathtt{r!a};\chi\oplus\mathtt{r!b};\chi))$ is

$$\mathtt{r!a};\mu\chi.(\mathtt{r!a};\chi\oplus\mathtt{r!b};\chi) \oplus \mathtt{r!b};\mu\chi.(\mathtt{r!a};\chi\oplus\mathtt{r!b};\chi)$$

Note that in this case, only one branch of the choice appears in $C\langle\mathbf{0}\rangle$, i.e., $\mathtt{r!a}$.

- In Example 3.9, we had

$$P \;=\; C\langle\mu\chi.C'\langle\chi\rangle\rangle \;=\; \mathtt{r!b};\mu\chi.\mathtt{r!b};\mathtt{r!a};\chi$$

We have $|\mathtt{n}| = \bot$ since

$$\begin{aligned}\mathtt{unfold}_1(\mu\chi.\mathtt{r!b};\mathtt{r!a};\chi) &= \mathtt{r!b};\mathtt{r!a};\mu\chi.\mathtt{r!b};\mathtt{r!a};\chi\\ &\neq \mathtt{r!b};\mu\chi.\mathtt{r!b};\mathtt{r!a};\chi\end{aligned}$$

Note that $\mathtt{r!b};\mathbf{0}$ is not a sub-tree of $\mathtt{r!b};\mathtt{r!a};\mathbf{0}$, and it is clear that unfolding the behaviour again will not help.

$\diamond$

Theorem 3.2 (Decidability)**.** *Typability is decidable.*

Proof. Decidability follows from the fact that the ready set of a system and the number of participants are finite. We show that the number of behaviour unfoldings needed to type

a system is also finite.

The need for unfolding occurs whenever a recursive participant interact with another participant, while not all the participants feature directly a recursive behaviour (cf. Examples 3.7 and 3.9). In this case, we need to unfold some participants (rule $[\equiv]$), then use rules $[\oplus]$, $[+]$, $[;]$, and/or $[\mathbf{0}]$ until rule $[\mu]$ is applicable. Note that rule $[\,|\,]$ empties the recursion environment, thus it cannot be used after a recursion definition if the axiom $[\chi]$ is to be used.

Consider the following system

$$S = S_0 \,|\, S_1$$

where

$$\begin{aligned} S_0 &= \mathtt{n}_1\big[C_1\langle\mu\chi.C_1'\langle\chi\rangle\rangle\big] \mid \ldots \mid \mathtt{n}_\mathtt{j}\big[C_j\langle\mu\chi.C_j'\langle\chi\rangle\rangle\big] \\ S_1 &= \mathtt{n}_{\mathtt{j}+1}\big[\mu\chi.C_{j+1}'\langle\chi\rangle\big] \mid \ldots \mid \mathtt{n}_\mathtt{k}[\mu\chi.C_{j+k}'\langle\chi\rangle] \end{aligned}$$

such that $\forall\, 1 \leqslant i \leqslant j :\ C_i\langle\_\rangle \neq \langle\_\rangle$, $S\!\updownarrow$, $S_0\!\not\updownarrow$ and $S_1\!\not\updownarrow$. The system S_0 consists of participants that do not exhibit a recursive behaviour (yet), while S_1 consists of participants exhibiting directly a recursive behaviour. There must be exactly one $\mathtt{n} \in \mathcal{P}(S)$ such that

$$S \equiv \mathtt{n}[S(\mathtt{n})] \,|\, S' \quad \text{and} \quad S'\!\not\updownarrow$$

Since if more that one pair of participants may interact directly after the recursion, it means that rule $[\,|\,]$ will have to be used further in the derivation tree, and therefore the recursion environment will be emptied.

Let C_i for $j < i \leqslant j+k$ be the empty context, we can rewrite S such that

$$S \equiv \big|_{i\in I}\, \mathtt{n}_\mathtt{i}\big[C_i\langle\mu\chi.C_i'\langle\chi\rangle\rangle\big] \qquad I = \{i \mid 1 \leqslant i \leqslant j+k\}$$

Given S as above, we will use the function $|\mathtt{n_i}|$ on each participant. Note that if one $|\mathtt{n_i}|$ is not defined, then S is not typable, cf. Example 3.9.

We define $M \stackrel{def}{=} \mathtt{max}\{|\mathtt{n_i}| \mid i \in I\}$, and $K(i) \stackrel{def}{=} M - |\mathtt{n_i}|$, in order to be able to unfold each behaviour so that all of them are unfolded to the same extent, let

$$S_* \equiv \big|_{i\in I}\, \mathtt{n_i}\big[\mathtt{unfold}_{K(i)}(C_i\langle \mu\chi.C_i'\langle\chi\rangle\rangle)\big]$$

We show that

$$\Gamma \vdash S_* \triangleright \mathcal{G}' \iff \Gamma \vdash S \triangleright \mathcal{G} \qquad \text{with } \mathcal{G} \equiv \mathcal{G}'$$

By definition of $\mathtt{unfold}(\_)$ and since $C\langle\_\rangle$ does not contain recursive definitions, we have

$$S_* \equiv \big|_{i\in I}\, \mathtt{n_i}\big[C_i\langle \mathtt{unfold}_{K(i)}(\mu\chi.C_i'\langle\chi\rangle)\rangle\big] \tag{3.4}$$

$$S_* \equiv \big|_{i\in I}\, \mathtt{n_i}\big[C_i\langle C_i'\langle C_i'\langle \ldots C_i'\langle \mu\chi.C_i'\langle\chi\rangle\rangle\ldots\rangle\rangle\big] \tag{3.5}$$

Where $C_i'\langle\_\rangle$ has been unfolded $K(i)$ times in (3.5). It is easy to see that S_* is typable if

$$\big|_{i\in I}\, \mathtt{n_i}\big[C_i\langle C_i'\langle C_i'\langle \ldots C_i'\langle \mathbf{0}\rangle\ldots\rangle\rangle\big] \quad \text{and} \quad \big|_{i\in I}\, \mathtt{n_i}\big[C_i'\langle\mathbf{0}\rangle\big]$$

are typable themselves, note that rule $[\equiv]$ does not need to be used to unfold the left-hand side system, since it is recursion free; and there is exactly one recursion less in the right hand side.

In fact, if we would unfold (3.4) once more, we would not get more chances to type S_*. Indeed, it would amount to add the right-hand sub-derivation to the left-hand sub-derivation. Thus, we have a bound on the number of required unfoldings and the type system is decidable. □

3.5.2 Uniqueness

Theorem 3.3 below shows that, given a typable system S, the global type that is assigned to it is unique up to the congruence rules from Section 3.3.

First, we give two lemmas that we will use in the proof of Theorem 3.3.

Lemma 3.5. *Let S be a program such that $S\not\updownarrow$, if $\Gamma \vdash S \triangleright \mathcal{G}$ then*

- $\mathcal{G} = \mathbf{0}$ *and* $\forall\, \mathtt{n} \in \mathcal{P}(S) :\ S(\mathtt{n}) = \mathbf{0}$*, or*
- $\mathcal{G} = \chi$ *and* $\forall\, \mathtt{n} \in \mathcal{P}(S) :\ S(\mathtt{n}) = \chi'$ *and* $\Gamma(\mathtt{n}, \chi') = \chi$

Proof. Since $S\not\updownarrow$, we must have that either (*i*) for each pair of participants $\mathtt{s}$ and $\mathtt{r}$ they do not have matching prefixes, or (*ii*) all the behaviours are of the form $\mathbf{0}$ or χ. In the former case, it will not be possible to reach one of the axioms (i.e., rules $[\mathbf{0}]$ or $[\chi]$). Thus, it contradicts the fact that the system is typable. In the latter case, if we have some participants with behaviour $\mathbf{0}$ and some with behaviours χ, then we would have to use rule $[\,|\,]$ to separate them in two subsystems, which implies that the environment Γ is emptied. Thus, we will not be able to reach the axioms for the subsystems consisting of processes with behaviour χ, and again we have a contradiction with the fact that S is typable. □

Lemma 3.6. *Let $\mathtt{n}[P + Q] \,|\, S$ be a program, if*

$$\Gamma \vdash \mathtt{r}[P + Q] \,|\, S \triangleright \mathcal{G} \quad \textit{and} \quad \Gamma \vdash \mathtt{r}[P] \,|\, S \triangleright \mathcal{G}$$

are derivable, then $\Gamma \vdash \mathtt{r}[Q] \,|\, S \triangleright \mathcal{G}$ is not derivable.

Proof. We show this by contradiction. Assume we have

$$\Gamma \vdash \mathtt{r}[P + Q] \,|\, S \triangleright \mathcal{G} \tag{3.6}$$

such that $P \neq \mathbf{0}$ and $Q \neq \mathbf{0}$, note that we cannot have $P = \chi$ (or $Q = \chi$), by definition of processes. Take $S\not\updownarrow$, if $S\updownarrow$, we can "wait" before applying rule $[+]$ until the condition

holds. Indeed, the process of $\mathtt{r}$ may only be affected by the derivation once one of the branches is discarded (rule $[+]$), then each of the branches P and Q may only be reduced if the rest of the system cannot "synchronise" without $\mathtt{r}$ (i.e., $S\not\updownarrow$ holds). Finally, recall that, by definition of processes, the process $P+Q$ synchronises with exactly one participant. Thus, we may "wait" until $\mathtt{r}$'s partner is ready to synchronise with $\mathtt{r}$.

Assume (by contradiction) that we have

$$\Gamma \vdash \mathtt{r}[P] \mid S \triangleright \mathcal{G} \qquad \text{and} \qquad \Gamma \vdash \mathtt{r}[Q] \mid S \triangleright \mathcal{G}'$$

We must have

$$\mathtt{r}[P] \mid S\updownarrow \quad \text{and} \quad \mathtt{r}[Q] \mid S\updownarrow$$

otherwise the systems would not be typable by Lemma 3.5, since we have $S\not\updownarrow$.

Take $P+Q \equiv \mathtt{s}?\mathtt{a};P_1 + \mathtt{s}?\mathtt{b};Q_1$. We reason by case analysis on the behaviour of $\mathtt{s}$ (up-to commutativity of internal choice).

- If we have

  $$S \equiv \mathtt{s}[\mathtt{r}!\mathtt{a};P_2 \oplus \mathtt{r}!\mathtt{b};Q_2] \mid S'$$

  From the judgement in (3.6) and rule $[\oplus]$, the two judgements below must be derivable

  $$\Gamma \vdash \mathtt{r}[P+Q] \mid \mathtt{s}[\mathtt{r}!\mathtt{a};P_2] \mid S' \triangleright \mathcal{G} \qquad \Gamma \vdash \mathtt{r}[P+Q] \mid \mathtt{s}[\mathtt{r}!\mathtt{b};Q_2] \mid S' \triangleright \mathcal{G}$$

  and we should also have

  $$\Gamma \vdash \mathtt{r}[\mathtt{s}?\mathtt{a};P_1] \mid \mathtt{s}[\mathtt{r}!\mathtt{a};P_2] \mid S' \triangleright \mathcal{G} \qquad \Gamma \vdash \mathtt{r}[\mathtt{s}?\mathtt{a};P_1] \mid \mathtt{s}[\mathtt{r}!\mathtt{b};Q_2] \mid S' \triangleright \mathcal{G}$$

  but the judgement on the right is not derivable since the prefix of the behaviours

of $\mathtt{r}$ and $\mathtt{s}$ do not match; and we have reached a contradiction. A dual reasoning shows that the Q branch is also not derivable.

- If we have

$$S \equiv \mathtt{s}[\mathtt{r}!\mathtt{b}; Q_2] \mid S'$$

then it contradicts the fact that $\Gamma \vdash \mathtt{r}[P] \mid S \blacktriangleright \mathcal{G}$ is derivable since the prefix $\mathtt{r}!\mathtt{b}$ will never be eliminated. Recall that all the message sorts guarding an external choice must distinct, thus no other branch than Q will permit to eliminate the prefix $\mathtt{r}!\mathtt{b}$.

- If we have

$$S \equiv \mathtt{s}[\mathtt{r}!\mathtt{a}; P_2 \oplus \ldots \mathtt{r}!\mathtt{b}; Q_2] \mid S'$$

where the ellipsis indicates a series of action that are not towards $\mathtt{r}$, i.e., $\mathtt{r}!\mathtt{b}$ is the first action towards $\mathtt{r}$ in the right-hand branch of the internal choice.

This also contradicts the fact that $\Gamma \vdash \mathtt{r}[P] \mid S \blacktriangleright \mathcal{G}$ is derivable since after all the prefixes in the right-hand branch of the internal choice have been eliminated, we return to the previous case. □

Theorem 3.3 (Unique typing)**.** *If* $\Gamma \vdash S \blacktriangleright \mathcal{G}$ *and* $\Gamma \vdash S \blacktriangleright \mathcal{G}'$ *then* $\mathcal{G} \equiv \mathcal{G}'$.

Proof. We show that each time a rule from Figure 3.3 is applicable, either no other rule is applicable, or the typing produces an equivalent global type.

- Due to their syntactic restrictions and the condition $S \not\downarrow$, the cases for rules $[;]$, $[\mu]$, $[\chi]$ and $[\mathbf{0}]$ are straightforward.

- The cases for rules $[\oplus]$ and $[\,|\,]$ follow naturally from the structural congruence rules of both systems and global types.

- The case for rule $[\equiv]$ follows from the fact that associativity and commutativity in S do not affect $\mathcal{G}$. In addition, if one unfolds a behaviour once more than required, we have the result since $\mu\chi.\mathcal{G} \equiv \mathcal{G}\{\mu\chi.\mathcal{G}/\chi\}$.

- For the case of rule $[+]$, we have to show that we cannot type a system with either branch of an external choice, i.e., only one branch allows the system to be typable. We show this case by contradiction, assume $\mathcal{G}_1 \not\equiv \mathcal{G}_2$,

$$[+]\ \frac{\begin{array}{c}\vdots\\ \hline \Gamma \vdash \mathtt{r}[P] \mid S \triangleright \mathcal{G}_1\end{array}}{\Gamma \vdash \mathtt{r}[P+Q] \mid S \triangleright \mathcal{G}_1} \qquad \text{and} \qquad [+]\ \frac{\begin{array}{c}\vdots\\ \hline \Gamma \vdash \mathtt{r}[Q] \mid S \triangleright \mathcal{G}_2\end{array}}{\Gamma \vdash \mathtt{r}[P+Q] \mid S \triangleright \mathcal{G}_2}$$

  By assumption, we have $\Gamma \vdash \mathtt{r}[P+Q] \mid S \triangleright \mathcal{G}_1$ or $\Gamma \vdash \mathtt{r}[P+Q] \mid S \triangleright \mathcal{G}_2$ and, by Lemma 3.6, we have a contradiction. □

3.5.3 Well-formedness and Projections

We characterise the relationships between the behaviours in a typable system and the projections of the global types that is assigned to it. In addition, we show that a synthesised global type is well-formed and projectable.

The following lemma shows a relationship between the ready set of a system and the ready set of its global type. We will use this lemma in several occasions.

Lemma 3.7. *If $\circ \vdash S \triangleright \mathcal{G}$ and $\mathcal{G}$ is well-formed and projectable, then*

$$\mathcal{G} \equiv (\mathtt{s} \to \mathtt{r} : \mathtt{a};\ \mathcal{G}_1 + \mathcal{G}_2) \mid \mathcal{G}_3 \iff S \equiv \mathtt{s}\big[\mathtt{r}!\mathtt{a}; P \oplus P'\big] \mid \mathtt{r}\big[\mathtt{s}?\mathtt{a}; Q + Q'\big] \mid S'$$

Proof. $(\Rightarrow)$ Assume that

$$\circ \vdash S \triangleright \mathcal{G} \quad \text{such that} \quad \mathcal{G} \equiv (\mathtt{s} \to \mathtt{r} : \mathtt{a};\ \mathcal{G}_1 + \mathcal{G}_2) \mid \mathcal{G}_3$$

is derivable. We show that either a rule introducing the corresponding operator is applicable or that an equivalent $\mathcal{G}$ can be inferred.

Assume $\mathcal{G} = \mathcal{G}' \mid \mathcal{G}_3$, if $\circ \vdash S \triangleright \mathcal{G}$ is derivable then we must have either $S \equiv S_1 \mid S_2$ such that

$$\circ \vdash S_1 \triangleright \mathcal{G}' \quad \text{and} \quad \circ \vdash S_2 \triangleright \mathcal{G}_3$$

are derivable (by rule [|]), or $S_3 \equiv \mathbf{0}$ and $\mathcal{G}_3 \equiv \mathbf{0}$. Observe that we must have $\mathcal{P}(S_1) \cap \mathcal{P}(S_2) = \varnothing$, by Lemma 3.4 and the fact that $\mathcal{G}$ is projectable.

Now, take $\mathcal{G}' = \mathcal{G}'' + \mathcal{G}_2$, we must have either

$$S_1 = \mathtt{s}\big[P_1 \oplus P_1'\big] \mid S_1'$$

such that

$$\circ \vdash \mathtt{s}[P_1] \mid S_1' \triangleright \mathcal{G}'' \quad \text{and} \quad \circ \vdash \mathtt{n}\big[P_1'\big] \mid S_1' \triangleright \mathcal{G}_2$$

are derivable (by rules [⊕] and $S_1' \not\updownarrow$), or $P_1' \equiv \mathbf{0}$ and $\mathcal{G}_2 \equiv \mathbf{0}$.

If we have $\mathcal{G}'' = \mathtt{s} \to \mathtt{r} : \mathtt{a}; \mathcal{G}_1$, we must have

$$P_1 \equiv \mathtt{r}!\mathtt{a}; P \quad \text{and} \quad S_1' \equiv \mathtt{r}\big[\mathtt{s}?\mathtt{a}; Q + Q'\big] \mid S_1''$$

and

$$\circ \vdash \mathtt{s}\big[\mathtt{r}!\mathtt{a}; P\big] \mid \mathtt{r}\big[\mathtt{s}?\mathtt{a}; Q + Q'\big] \mid S_1'' \triangleright \mathtt{s} \to \mathtt{r} : \mathtt{a}; \mathcal{G}_1$$

derivable.

Finally, considering the systems S_1 and S_2, and the processes of $\mathtt{s}$ and $\mathtt{r}$, we have the required result:

$$S \equiv \mathtt{s}\big[\mathtt{r}!\mathtt{a}; P \oplus P'\big] \mid \mathtt{r}\big[\mathtt{s}?\mathtt{a}; Q + Q'\big] \mid S'$$

($\Leftarrow$) Assume

$$\circ \vdash \mathtt{s}\big[\mathtt{r}!\mathtt{a}; P \oplus P'\big] \mid \mathtt{r}\big[\mathtt{s}?\mathtt{a}; Q + Q'\big] \mid S' \triangleright \mathcal{G}$$

We show that either a rule introducing the corresponding operator is applicable or that an

equivalent $\mathcal{G}$ can be inferred. Either there is S_1 and S_2 such that $S' \equiv S_1 \mid S_2$, and

$$\circ \vdash \mathtt{s}\big[\mathtt{r}!\mathtt{a}\,;P \oplus P'\big] \mid \mathtt{r}\big[\mathtt{s}?\mathtt{a}\,;Q+Q'\big] \mid S_1 \blacktriangleright \mathcal{G}' \quad \text{and} \quad \circ \vdash S_2 \blacktriangleright \mathcal{G}_3$$

or $\mathcal{G}_3 \equiv \mathbf{0}$. Note that we can assume that $S_1 \not\downarrow^{\checkmark}$ since one could apply rule [|] as many times as necessary until it holds.

Now, either there is $\mathcal{G}' \equiv \mathcal{G}'' + \mathcal{G}_2$ such that

$$\circ \vdash \mathtt{s}[\mathtt{r}!\mathtt{a}\,;P] \mid \mathtt{r}\big[\mathtt{s}?\mathtt{a}\,;Q+Q'\big] \mid S_1 \blacktriangleright \mathcal{G}''$$

and

$$\circ \vdash \mathtt{s}\big[P'\big] \mid \mathtt{r}\big[\mathtt{s}?\mathtt{a}\,;Q+Q'\big] \mid S_1 \blacktriangleright \mathcal{G}_2$$

or $\mathcal{G}_2 = \mathbf{0}$.

Finally, for the judgement

$$\circ \vdash \mathtt{s}[\mathtt{r}!\mathtt{a}\,;P] \mid \mathtt{r}\big[\mathtt{s}?\mathtt{a}\,;Q+Q'\big] \mid S_1 \blacktriangleright \mathcal{G}''$$

to be derivable, we must have $\mathcal{G}'' \equiv \mathtt{s} \to \mathtt{r} : \mathtt{a}\,; \mathcal{G}_1$. Considering $\mathcal{G}'$ and $\mathcal{G}''$, we have the required result. □

We show that there is a correspondence between the behaviours of the original system and the projections of its synthesised global type. The behaviour of a participant in S is a simulation of the projection of a synthesised global type from S onto the participant. Intuitively, the other direction is lost due to rule [+], indeed external choice branches which are never chosen are not "recorded" in the synthesised global type.

Example 3.12. Consider the following system

$$S \equiv \mathtt{s}[\mathtt{r}!\mathtt{a}] \mid \mathtt{r}[\mu\chi.(\mathtt{s}?\mathtt{b}\,;\chi + \mathtt{s}?\mathtt{a})]$$

We show that S has global type $\mathtt{s} \to \mathtt{r} : \mathtt{a}$, via the derivation below.

$$\dfrac{\dfrac{\dfrac{}{\circ \vdash \mathtt{s}[\mathbf{0}] \mid \mathtt{r}[\mathbf{0}] \triangleright \mathbf{0}}\,[\mathbf{0}]}{\circ \vdash \mathtt{s}[\mathtt{r}!\mathtt{a}] \mid \mathtt{r}[\mathtt{s}?\mathtt{a}] \triangleright \mathtt{s} \to \mathtt{r} : \mathtt{a}}\,[;]}{\circ \vdash \mathtt{s}[\mathtt{r}!\mathtt{a}] \mid \mathtt{r}[\mathtt{s}?\mathtt{b}; (\mu\chi.(\mathtt{s}?\mathtt{b}; \chi + \mathtt{s}?\mathtt{a})) + \mathtt{s}?\mathtt{a}] \triangleright \mathtt{s} \to \mathtt{r} : \mathtt{a}}\,[+]$$
$$\dfrac{}{\circ \vdash S \triangleright \mathtt{s} \to \mathtt{r} : \mathtt{a}}\,[\equiv]$$

$\diamond$

Note that, given a participant with an external choice, it must be the case that at least one branch of the external choice is matched by a corresponding sending action. For instance, in Example 3.12, we can only eliminate the prefix $\mathtt{s}?\mathtt{a}$ via rule [;].

We now give our notion of simulation, which preserves sending actions in both directions; it requires all the receiving actions to be preserved in one direction and the *existence* of a matching receiving action in the other direction.

Definition 3.8 ($\lesssim$)**.** *$P \lesssim Q$ if and only if*

- *$P \xrightarrow{\mathtt{r}!\mathtt{a}} P'$ implies that there is Q' such that $Q \xrightarrow{\mathtt{r}!\mathtt{a}} Q'$ and $P' \lesssim Q'$,*
- *$Q \xrightarrow{\mathtt{r}!\mathtt{a}} Q'$ implies that there is P' such that $P \xrightarrow{\mathtt{r}!\mathtt{a}} P'$ and $P' \lesssim Q'$,*
- *$P \xrightarrow{\mathtt{s}?\mathtt{a}} P'$ implies that there is Q' such that $Q \xrightarrow{\mathtt{s}?\mathtt{a}} Q'$ and $P' \lesssim Q'$, and*
- *$Q \xrightarrow{\mathtt{s}?\mathtt{a}} Q'$ implies that there is $\mathtt{b}$, Q'', and P' such that $Q \xrightarrow{\mathtt{s}?\mathtt{b}} Q''$, $P \xrightarrow{\mathtt{s}?\mathtt{b}} P'$, and $P' \lesssim Q''$,*

Definition 3.8 implies naturally the following proposition.

Proposition 3.1. *Let Q be a process. If $\mathbf{0} \lesssim Q$ then $Q \equiv \mathbf{0}$.*

Proof. By contradiction. If $Q = \bigoplus_{i\in I} \mathtt{r}_i!\mathtt{a}_i;Q_i$, then we reach a contradiction since $\mathbf{0}$ cannot match any of the sending actions. If $Q = \sum_{i\in I} \mathtt{s}?\mathtt{a}_i;Q_i$, then there should exists at least one action that is matched by $\mathbf{0}$, which leads to a contradiction. □

Lemma 3.8. *If* $\circ \vdash \mathtt{n}[P] \mid S \triangleright \mathcal{G}$ *then* $\mathcal{G}\upharpoonright_{\mathtt{n}} \lesssim P$.

Proof. The result follows directly from following intermediary statement (since we consider only closed behaviours).

Let $S' \equiv \mathtt{n}[P] \mid S$, $\mathcal{G}$, and Γ such that

- $\forall \mathtt{n}' \in \mathcal{P}(S') : \forall \chi \in \mathtt{fv}(S'(\mathtt{n}')) : \exists \chi' : \Gamma(\mathtt{n}', \chi) = \chi'$
- $\forall \chi \in \mathtt{fv}(\mathcal{G}) : \forall \mathtt{n}' \in \mathcal{P}(S') : \exists \chi' : \Gamma(\mathtt{n}', \chi') = \chi$

if $\Gamma \vdash \mathtt{n}[P] \mid S \triangleright \mathcal{G}$ then $\mathcal{G}\upharpoonright_{\mathtt{n}} \lesssim P$.

The proof is by case analysis on the actions enabled for P and $\mathcal{G}\upharpoonright_{\mathtt{n}}$ (we treat the recursive cases separately), then by induction on the structure of the derivation.

***P* sends.** If $P \xrightarrow{\mathtt{r}!\mathtt{a}} P'$, then we must have $P \equiv \mathtt{r}!\mathtt{a};P' \oplus P''$. Assume we have $S \equiv \mathtt{r}[\mathtt{n}?\mathtt{a};Q + Q'] \mid S''$ and $S'' \not\xrightarrow{\tau}$, if it is not the case, we may apply other rules until we obtain such a system. Indeed, the derivation may not affect the process of $\mathtt{n}$ until the system cannot synchronise without $\mathtt{n}$.[2]

$$
\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash \mathtt{n}[P'] \mid \mathtt{r}[Q] \mid S'' \triangleright \mathcal{G}_1}\,[;]}{\Gamma \vdash \mathtt{n}[\mathtt{r}!\mathtt{a};P'] \mid \mathtt{r}[\mathtt{n}?\mathtt{a};Q] \mid S'' \triangleright \mathtt{n}\to\mathtt{r}:\mathtt{a};\, \mathcal{G}_1}\,[+] \qquad \dfrac{\vdots}{\Gamma \vdash \mathtt{n}[P''] \mid \mathtt{r}[\mathtt{n}?\mathtt{a};Q + Q'] \mid S'' \triangleright \mathcal{G}_2}}{\Gamma \vdash \mathtt{n}[\mathtt{r}!\mathtt{a};P' \oplus P''] \mid \mathtt{r}[\mathtt{n}?\mathtt{a};Q + Q'] \mid S'' \triangleright \mathtt{n}\to\mathtt{r}:\mathtt{a};\, \mathcal{G}_1 + \mathcal{G}_2}\,[\oplus]
$$

Thus, we have $\mathcal{G}\upharpoonright_{\mathtt{n}} \xrightarrow{\mathtt{r}!\mathtt{a}} \mathcal{G}_1\upharpoonright_{\mathtt{n}}$, and, by induction hypothesis, we have

$$\mathcal{G}_1\upharpoonright_{\mathtt{n}} \lesssim P' \quad \text{and} \quad \mathcal{G}_2\upharpoonright_{\mathtt{n}} \lesssim P''$$

[2] Except for rule [≡], but it may only fold or unfold the process.

as required.

P **receives.** If $P \xrightarrow{\mathtt{s?a}} P'$, then we must have $P \equiv \mathtt{s?a}\,;P' + P''$. There are two cases: either $P'' \equiv \mathbf{0}$ or $P'' \not\equiv \mathbf{0}$.

- If $P'' \equiv \mathbf{0}$, then we must have a matching action for $\mathtt{s?a}$ (we cannot apply rule [+]). Assume we have $S \equiv \mathtt{s}[\mathtt{n!a}\,;Q] \mid S''$ and $S'' \not\downarrow$, if it is not the case, we may apply other rules until we obtain such a system (as above). We have the following derivation:

$$[;]\ \frac{\dfrac{\vdots}{\Gamma \vdash \mathtt{n}\big[P'\big] \mid \mathtt{s}[Q] \mid S'' \triangleright \mathcal{G}'}}{\Gamma \vdash \mathtt{n}\big[\mathtt{s?a}\,;P'\big] \mid \mathtt{s}[\mathtt{n!a}\,;Q] \mid S'' \triangleright \mathtt{s} \to \mathtt{n}:\mathtt{a};\ \mathcal{G}'}$$

  Thus, we have $\mathcal{G}\upharpoonright_{\mathtt{n}} \xrightarrow{\mathtt{s?a}} \mathcal{G}'\upharpoonright_{\mathtt{n}}$, and, by induction hypothesis, we have $\mathcal{G}'\upharpoonright_{\mathtt{n}} \lesssim P'$ as required.

- If $P'' \not\equiv \mathbf{0}$ then there are two sub-cases:
  - Either there is matching action for $\mathtt{s?a}$, in which case we obtain the result by discarding the P'' branch with rule [+], then applying the same reasoning as above to apply rule [;]; or
  - there is not matching action for $\mathtt{s?a}$, in which case we can discard the $\mathtt{s?a}\,;P'$ branch with rule [+]. Since $P'' \not\equiv \mathbf{0}$ and the system is typable, there must be one branch of $\mathtt{n}$ that will be recorded in $\mathcal{G}$, i.e., one cannot discard the "last" branch of an external choice (cf. the case where $P'' \equiv \mathbf{0}$). The result follows from the fact that *there exists* one branch of the external choice that will be "recorded" in $\mathcal{G}$.

$\mathcal{G}\upharpoonright_{\mathtt{n}}$ **sends.** If $\mathcal{G}\upharpoonright_{\mathtt{n}} \xrightarrow{\mathtt{r!a}} \mathcal{G}_1\upharpoonright_{\mathtt{n}}$, then we must have $\mathcal{G}\upharpoonright_{\mathtt{n}} \equiv \mathtt{r!a}\,;\mathcal{G}_1\upharpoonright_{\mathtt{n}} \oplus \mathcal{G}_2\upharpoonright_{\mathtt{n}}$. We must have the following sub-derivation of $\Gamma \vdash \mathtt{n}[P] \mid S \triangleright \mathcal{G}$ such that $P \equiv \mathtt{r!a}\,;P' \oplus P''$ where

all the previous steps of the derivation did not affect $\mathtt{n}[P]$ (otherwise we would not have $\mathcal{G}\upharpoonright_{\mathtt{n}} \equiv \mathtt{r}!\mathtt{a};\mathcal{G}_1\upharpoonright_{\mathtt{n}} \oplus \mathcal{G}_2\upharpoonright_{\mathtt{n}}$). Note that we apply rules [⊕] and [+] in one step, for simplicity.

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash \mathtt{n}[P'] \mid \mathtt{r}[Q] \mid S'' \triangleright \mathcal{G}_1}[;]}{\Gamma \vdash \mathtt{n}[\mathtt{r}!\mathtt{a};P'] \mid \mathtt{r}[\mathtt{s}?\mathtt{a};Q] \mid S'' \triangleright \mathtt{n}\to\mathtt{r}:\mathtt{a};\mathcal{G}_1} \qquad \dfrac{\vdots}{\Gamma \vdash \mathtt{n}[P''] \mid \mathtt{r}[Q'] \mid S'' \triangleright \mathcal{G}_2}}{\Gamma \vdash \mathtt{n}[\mathtt{r}!\mathtt{a};P' \oplus P''] \mid \mathtt{r}[\mathtt{s}?\mathtt{a};Q+Q'] \mid S'' \triangleright \mathtt{n}\to\mathtt{r}:\mathtt{a};\mathcal{G}_1+\mathcal{G}_2}[\oplus][+]$$

We have $P \xrightarrow{\mathtt{r}!\mathtt{a}} P'$, and, by induction hypothesis, we have

$$\mathcal{G}_1\upharpoonright_{\mathtt{n}} \lesssim P' \quad \text{and} \quad \mathcal{G}_2\upharpoonright_{\mathtt{n}} \lesssim P''$$

as required.

$\mathcal{G}\upharpoonright_{\mathtt{n}}$ **receives.** If $\mathcal{G}\upharpoonright_{\mathtt{n}} \xrightarrow{\mathtt{s}?\mathtt{a}} \mathcal{G}_1\upharpoonright_{\mathtt{n}}$, then we must have $\mathcal{G}\upharpoonright_{\mathtt{n}} \equiv \mathtt{s}?\mathtt{a};\mathcal{G}_1\upharpoonright_{\mathtt{n}}$.[3] We must have the following sub-derivation of $\Gamma \vdash \mathtt{n}[P] \mid S \triangleright \mathcal{G}$ such that $P \equiv \mathtt{s}?\mathtt{a};P' + P''$ where all the previous steps of the derivation did not affect $\mathtt{n}[P]$ (otherwise we would not have $\mathcal{G}\upharpoonright_{\mathtt{n}} \equiv \mathtt{s}?\mathtt{a};\mathcal{G}_1\upharpoonright_{\mathtt{n}}$).

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash \mathtt{n}[P'] \mid \mathtt{s}[Q] \mid S'' \triangleright \mathcal{G}_1}[;]}{\Gamma \vdash \mathtt{n}[\mathtt{s}?\mathtt{a};P'] \mid \mathtt{s}[\mathtt{n}!\mathtt{a};Q] \mid S'' \triangleright \mathtt{s}\to\mathtt{n}:\mathtt{a};\mathcal{G}_1}[+]}{\Gamma \vdash \mathtt{n}[\mathtt{s}?\mathtt{a};P' + P''] \mid \mathtt{s}[\mathtt{n}!\mathtt{a};Q] \mid S'' \triangleright \mathtt{s}\to\mathtt{n}:\mathtt{a};\mathcal{G}_1}$$

We have $P \xrightarrow{\mathtt{s}?\mathtt{a}} P'$, and, by induction hypothesis, we have $\mathcal{G}_1\upharpoonright_{\mathtt{n}} \lesssim P'$ as required.

Recursion in projection. If $\mathcal{G}\upharpoonright_{\mathtt{n}} \equiv \mu\chi.\mathcal{G}_1\upharpoonright_{\mathtt{n}}$, then we must have used rule [μ], thus we must have $P \equiv \mu\chi.P'$, and $\forall \mathtt{m} \in \mathcal{P}(S) :\ S(\mathtt{m}) \equiv \mu\chi.Q$ and the rest follows by induction

[3] We abstract from possible external choice.

hypothesis and the fact that all participants must be added in Γ together with their local recursion variable.

Recursion in local type. If $P \equiv \mu\chi.P'$, then there are two cases either rule $[\mu]$ is the last rule applied, in which case, the result follows as above. Otherwise $[\equiv]$ is the last rule that has been applied, and we must have the following sub-derivation:

$$[\equiv]\ \frac{\Gamma \vdash \mathtt{n}\big[P'\{\mu\chi.P'/\chi\}\big] \mid S \triangleright \mathcal{G}}{\Gamma \vdash \mathtt{n}\big[\mu\chi.P'\big] \mid S \triangleright \mathcal{G}}$$

By induction hypothesis, we have $\mathcal{G}\!\upharpoonright_{\mathtt{n}} \lesssim P'\{\mu\chi.P'/\chi\}$, and the result follows from the rule

$$P'\{\mu\chi.P'/\chi\} \equiv \mu\chi.P' \qquad \square$$

Theorem 3.4 (Well-formedness)**.** *If $\Gamma \vdash S \triangleright \mathcal{G}$ then $\vdash \mathcal{G}$ and $\mathcal{G}$ is projectable.*

Proof. The proof is by induction on the derivation $\Gamma \vdash S \triangleright \mathcal{G}$. We make a case analysis on the last rule used.

Case $[\chi]$**.** Trivial by Definition 3.1.

Cases $[\mathbf{0}]$: Trivial by Definition 3.1.

Cases $[\equiv]$: Trivial by induction hypothesis.

Case $[:]$**.** We have $\Gamma \vdash \mathtt{s}[P] \mid \mathtt{r}[Q] \mid S' \triangleright \mathcal{G}'$,

$$\mathcal{G} = \mathtt{s} \to \mathtt{r} : \mathtt{a};\, \mathcal{G}', \qquad S = \mathtt{s}[\mathtt{r}!\mathtt{a}; P] \mid \mathtt{r}[\mathtt{s}?\mathtt{a}; Q] \mid S', \quad \text{and} \quad S' \not\downarrow$$

- *WF.* We show that we have

$$\forall\, \mathtt{n}_1 \to \mathtt{n}_2 : \_ \in \mathtt{R}(\mathcal{G}') :\ \{\mathtt{s}, \mathtt{r}\} \cap \{\mathtt{n}_1, \mathtt{n}_2\} \neq \varnothing$$

by contradiction. By induction hypothesis, we know that

$$\Gamma \vdash \mathtt{s}[P] \mid \mathtt{r}[Q] \mid S' \blacktriangleright \mathcal{G}'$$

If we had $\mathcal{G}' \equiv (\mathtt{n}_1 \to \mathtt{n}_2 : \mathtt{b};\ \mathcal{G}_0 + \mathcal{G}_1) \mid \mathcal{G}_2$ with $\mathtt{n}_\mathtt{i} \neq \mathtt{s}$ and $\mathtt{n}_\mathtt{i} \neq \mathtt{r}$ and $\mathtt{i} \in \{1,2\}$, by Lemma 3.7, we would have

$$S' \equiv \mathtt{n}_1\big[\mathtt{n}_2!\mathtt{b}; P'_0 \oplus P'_1\big] \mid \mathtt{n}_2\big[\mathtt{n}_1?\mathtt{b}; Q'_0 + Q'_1\big] \mid S''$$

which is in contradiction with the premise $S'\not\downarrow$.

- *Projection.* By induction hypothesis, we know that $\mathcal{G}'\!\upharpoonright_\mathtt{n}$ is defined for all $\mathtt{n} \in \mathcal{P}(\mathcal{G}')$. By Definition 3.1, we have that $\mathcal{G}\!\upharpoonright_\mathtt{s} = \mathtt{r}!\mathtt{a}; \mathcal{G}'\!\upharpoonright_\mathtt{s}$, $\mathcal{G}\!\upharpoonright_\mathtt{r} = \mathtt{s}?\mathtt{a}; \mathcal{G}'\!\upharpoonright_\mathtt{r}$, and $\mathcal{G}\!\upharpoonright_\mathtt{n} = \mathcal{G}'\!\upharpoonright_\mathtt{n}$, for $\mathtt{s} \neq \mathtt{n} \neq \mathtt{r}$.

Case $[\oplus]$**.** We have

$$\mathcal{G} = \mathcal{G}_0 + \mathcal{G}_1, \qquad S = \mathtt{s}[P \oplus Q] \mid S' \quad \text{and} \quad S'\not\downarrow$$

- *WF.* We have to show that

$$\forall\, \mathtt{s} \to \mathtt{r} : \mathtt{a} \in \mathtt{R}(\mathcal{G}_0) :\ \forall\, \mathtt{s}' \to \mathtt{r}' : \mathtt{b} \in \mathtt{R}(\mathcal{G}_1) :\ \mathtt{s} = \mathtt{s}' \ \wedge\ (\mathtt{r}, \mathtt{a}) \neq (\mathtt{r}', \mathtt{b})$$

$(\mathtt{r}, \mathtt{a}) \neq (\mathtt{r}', \mathtt{b})$ follows directly from the syntax of processes, i.e., we have

$$P \equiv \bigoplus_{i \in I} \mathtt{r}_\mathtt{i}!\mathtt{a}_\mathtt{i}; P_i \quad \text{and} \quad Q \equiv \bigoplus_{j \in J} \mathtt{r}_\mathtt{j}!\mathtt{a}_\mathtt{j}; P_j \quad \text{and} \quad I \cap J = \varnothing$$

$$\forall\, i, j \in I \cup J :\ i \neq j \implies (\mathtt{r}_\mathtt{i}, \mathtt{a}_\mathtt{i}) \neq (\mathtt{r}_\mathtt{j}, \mathtt{a}_\mathtt{j})$$

We have to show that for all prefixes in $\mathcal{G}_0$ and $\mathcal{G}_1$, $\mathtt{s}$ is the sender. In other words,

the participant who makes the internal choice at the global type level must be the same in all the branches. By contradiction, assume we have

$$\mathcal{G}_0 = \mathtt{s} \to \mathtt{r} : \mathtt{a} + \mathtt{s} \to \mathtt{r} : \mathtt{b} \quad \text{and} \quad \mathcal{G}_1 = \mathtt{s}' \to \mathtt{r}'' : \mathtt{b}' \quad \text{and} \quad \mathtt{s} \neq \mathtt{s}'$$

then we must have a system of the form

$$S \equiv \mathtt{s}\big[\mathtt{r}!\mathtt{a} \oplus \mathtt{r}'!\mathtt{b}\big] \mid \mathtt{s}'\big[\mathtt{r}''!\mathtt{b}'\big] \mid \mathtt{r}\big[\mathtt{s}?\mathtt{a} + \mathtt{s}?\mathtt{b}\big] \mid \mathtt{r}''\big[\mathtt{s}'?\mathtt{b}'\big] \mid S''$$

which is in contradiction with the premise $S' \not\downarrow$.

By the result above and $\vdash \mathcal{G}_0$ and $\vdash \mathcal{G}_1$ by induction hypothesis, we have $\vdash \mathcal{G}$.

- *Projection.* If $\mathtt{n}$ is the sender, we have the result directly by induction hypothesis, i.e., both $\mathcal{G}_0 \upharpoonright_{\mathtt{n}}$ and $\mathcal{G}_1 \upharpoonright_{\mathtt{n}}$ are defined by induction hypothesis, thus so is $\mathcal{G}_0 \upharpoonright_{\mathtt{n}} \oplus \mathcal{G}_1 \upharpoonright_{\mathtt{n}}$.

  For all $\mathtt{n}$ not the sender in $\mathcal{G}_0$ and $\mathcal{G}_1$, we have to show that, knowing by induction hypothesis that both $\mathcal{G}_0 \upharpoonright_{\mathtt{n}}$ and $\mathcal{G}_1 \upharpoonright_{\mathtt{n}}$ are defined,

  $$\mathcal{G}_0 \upharpoonright_{\mathtt{n}} \uplus \mathcal{G}_1 \upharpoonright_{\mathtt{n}} \neq \bot$$

  We show this by contradiction, if $\_ \uplus \_$ is undefined, it may only originates from the following cases, by Definition 3.2 (we consider only the prefixes of projections, without loss of generality).

  Send. If we have[4]

  $$\mathcal{G}_0 \upharpoonright_{\mathtt{n}} \equiv \mathtt{r}!\mathtt{a} \quad \text{and} \quad \mathcal{G}_1 \upharpoonright_{\mathtt{n}} \equiv \mathtt{r}!\mathtt{b}$$

  Take the system of the form below from which the projections above must originate,

[4]This case corresponds to the case where the behaviour of $\mathtt{n}$ is different in two branches, while being unaware of which branch was chosen: $\mathtt{n}$ has to send either $\mathtt{a}$ or $\mathtt{b}$ depending on which branch was chosen by $\mathtt{s}$ – without knowing which branch was selected.

by Lemma 3.8 and definition of projections.

$$S \equiv \mathtt{s}[\mathtt{r}!\mathtt{a} \oplus \mathtt{r}!\mathtt{b}] \mid \mathtt{r}[\mathtt{s}?\mathtt{a}\,;\mathtt{n}?\mathtt{a} + \mathtt{s}?\mathtt{b}\,;\mathtt{n}?\mathtt{b}] \mid \mathtt{n}[\mathtt{r}!\mathtt{a} \oplus \mathtt{r}!\mathtt{b}]$$

we must have the following derivation

$$\dfrac{\dfrac{\dfrac{\dfrac{\nabla_l}{\Gamma \vdash \mathtt{s}[\mathbf{0}] \mid \mathtt{r}[\mathtt{n}?\mathtt{a}] \mid \mathtt{n}[\ldots] \triangleright \ldots}\,[\oplus]}{\Gamma \vdash \mathtt{s}[\mathtt{r}!\mathtt{a}] \mid \mathtt{r}[\mathtt{s}?\mathtt{a}\,;\mathtt{n}?\mathtt{a}] \mid \mathtt{n}[\ldots] \triangleright \mathcal{G}_0}\,[;]}{\Gamma \vdash \mathtt{s}[\mathtt{r}!\mathtt{a}] \mid \mathtt{r}[\ldots] \mid \mathtt{n}[\ldots] \triangleright \mathcal{G}_0}\,[+] \qquad \dfrac{\dfrac{\dfrac{\dfrac{\nabla_r}{\Gamma \vdash \mathtt{s}[\mathbf{0}] \mid \mathtt{r}[\mathtt{n}?\mathtt{b}] \mid \mathtt{n}[\ldots] \triangleright \ldots}\,[\oplus]}{\Gamma \vdash \mathtt{s}[\mathtt{r}!\mathtt{b}] \mid \mathtt{r}[\mathtt{s}?\mathtt{b}\,;\mathtt{n}?\mathtt{b}] \mid \mathtt{n}[\ldots] \triangleright \mathcal{G}_1}\,[;]}{\Gamma \vdash \mathtt{s}[\mathtt{r}!\mathtt{b}] \mid \mathtt{r}[\ldots] \mid \mathtt{n}[\ldots] \triangleright \mathcal{G}_1}\,[+]}{\Gamma \vdash S \triangleright \mathcal{G}}\,[\oplus]$$

Where ∇_l is as follows

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash \mathtt{s}[\mathbf{0}] \mid \mathtt{r}[\mathbf{0}] \mid \mathtt{n}[\mathbf{0}] \triangleright \ldots}\,[\mathbf{0}]}{\Gamma \vdash \mathtt{s}[\mathbf{0}] \mid \mathtt{r}[\mathtt{n}?\mathtt{a}] \mid \mathtt{n}[\mathtt{r}!\mathtt{a}] \triangleright \ldots}\,[;] \qquad \dfrac{\bot}{\Gamma \vdash \mathtt{s}[\mathbf{0}] \mid \mathtt{r}[\mathtt{n}?\mathtt{a}] \mid \mathtt{n}[\mathtt{r}!\mathtt{b}] \triangleright \ldots}}{\Gamma \vdash \mathtt{s}[\mathbf{0}] \mid \mathtt{r}[\mathtt{n}?\mathtt{a}] \mid \mathtt{n}[\ldots] \triangleright \ldots}\,[\oplus]$$

Clearly, we have a contradiction with the hypothesis that $\mathcal{G}$ is derivable. Note that ∇_r is similar to ∇_l.

Receive. If we have

$$\mathcal{G}_0 \upharpoonright_{\mathtt{n}} \equiv \mathtt{r}?\mathtt{a} + \mathtt{r}?\mathtt{b} \quad \text{and} \quad \mathcal{G}_1 \upharpoonright_{\mathtt{n}} \equiv \mathtt{r}?\mathtt{b}$$

Take the system of the form below from which the projections above must originate,

by Lemma 3.8 and definition of projections.

$$S \equiv \mathtt{s}[\mathtt{r}!\mathtt{a} \oplus \mathtt{r}!\mathtt{b}] \mid \mathtt{r}[\mathtt{s}?\mathtt{a};\mathtt{n}!\mathtt{a} + \mathtt{s}?\mathtt{b};\mathtt{n}!\mathtt{b}] \mid \mathtt{n}[\mathtt{r}?\mathtt{a} + \mathtt{r}?\mathtt{b}]$$

We must have the following derivation

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash \mathtt{s}[\mathbf{0}] \mid \mathtt{r}[\mathbf{0}] \mid \mathtt{n}[\mathbf{0}] \triangleright \mathbf{0}}[\mathbf{0}]}{\Gamma \vdash \mathtt{s}[\mathbf{0}] \mid \mathtt{r}[\mathtt{n}!\mathtt{a}] \mid \mathtt{n}[\mathtt{r}!\mathtt{a}] \triangleright \mathcal{G}_0'}[;]}{\Gamma \vdash \mathtt{s}[\mathbf{0}] \mid \mathtt{r}[\mathtt{n}!\mathtt{a}] \mid \mathtt{n}[\ldots] \triangleright \mathcal{G}_0'}[+]}{\Gamma \vdash \mathtt{s}[\mathtt{r}!\mathtt{a}] \mid \mathtt{r}[\mathtt{s}?\mathtt{a};\mathtt{n}!\mathtt{a}] \mid \mathtt{n}[\ldots] \triangleright \mathcal{G}_0}[;] \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash \mathtt{s}[\mathbf{0}] \mid \mathtt{r}[\mathbf{0}] \mid \mathtt{n}[\mathbf{0}] \triangleright \mathbf{0}}[\mathbf{0}]}{\Gamma \vdash \mathtt{s}[\mathbf{0}] \mid \mathtt{r}[\mathtt{n}!\mathtt{b}] \mid \mathtt{n}[\mathtt{r}!\mathtt{b}] \triangleright \mathcal{G}_1'}[;]}{\Gamma \vdash \mathtt{s}[\mathbf{0}] \mid \mathtt{r}[\mathtt{n}!\mathtt{b}] \mid \mathtt{n}[\ldots] \triangleright \mathcal{G}_1'}[+]}{\Gamma \vdash \mathtt{s}[\mathtt{r}!\mathtt{b}] \mid \mathtt{r}[\mathtt{s}?\mathtt{b};\mathtt{n}!\mathtt{b}] \mid \mathtt{n}[\ldots] \triangleright \mathcal{G}_1}[+]}{\Gamma \vdash \mathtt{s}[\mathtt{r}!\mathtt{a}] \mid \mathtt{r}[\ldots] \mid \mathtt{n}[\ldots] \triangleright \mathcal{G}_0} \qquad \dfrac{\Gamma \vdash \mathtt{s}[\mathtt{r}!\mathtt{b}] \mid \mathtt{r}[\ldots] \mid \mathtt{n}[\ldots] \triangleright \mathcal{G}_1}{\Gamma \vdash S \triangleright \mathcal{G}}[\oplus]$$

where

$$\mathcal{G}_0 = \mathtt{s} \to \mathtt{r} : \mathtt{a};\ \mathcal{G}_0' \quad \text{and} \quad \mathcal{G}_0' = \mathtt{r} \to \mathtt{n} : \mathtt{a};\ \mathbf{0}$$

and

$$\mathcal{G}_1 = \mathtt{s} \to \mathtt{r} : \mathtt{b};\ \mathcal{G}_1' \quad \text{and} \quad \mathcal{G}_1' = \mathtt{r} \to \mathtt{n} : \mathtt{b};\ \mathbf{0}$$

Clearly, we have that $\mathcal{G}_0 \upharpoonright_{\mathtt{n}} \not\equiv \mathtt{r}?\mathtt{a} + \mathtt{r}?\mathtt{b}$, which is a contradiction.

Send-Receive. The case where one projection is an internal choice, while the other is an external choice, is discarded by the syntax of processes.

Case [+].

- *WF.* By induction hypothesis, we have $\vdash \mathcal{G}$.
- *Projection.* By induction hypothesis.

Case $[\,|\,]$**.** We have

$$\mathcal{G} = \mathcal{G}_0 \mid \mathcal{G}_1$$

- *WF.* We have to show that

  $$\mathcal{P}(\mathcal{G}_0) \cap \mathcal{P}(\mathcal{G}_1) = \varnothing$$

  By definition of systems, we know that there cannot be two participants with the same name in a system, since $S \mid S'$ is a system we have that

  $$\mathcal{P}(S) \cap \mathcal{P}(S') = \varnothing$$

  and by Lemma 3.4, we have $\mathcal{P}(\mathcal{G}_0) \cap \mathcal{P}(\mathcal{G}_1) = \varnothing$.

- *Projection.* For all $\mathtt{n} \in \mathcal{P}(\mathcal{G})$, $\mathcal{G}\upharpoonright_{\mathtt{n}}$ is defined by induction hypothesis.

Case $[\mu]$**.** We have $\mathcal{G} = \mu\chi.\mathcal{G}'$,

$$S = \mathtt{n}_1[\mu X_1.P_1] \mid \ldots \mid \mathtt{n}_k[\mu X_k.P_k] \quad \text{and} \quad \exists\, 1 \leqslant i, j \leqslant k :\ (\mathtt{n}_i[P_i] \mid \mathtt{n}_j[P_j])\!\updownarrow$$

- *WF.* We have to show that

  $$\chi \in \mathtt{fv}(\mathcal{G}') \Rightarrow |\mathtt{Indep}(\mathcal{G}')| = 1$$

  which follows from the fact that the context Γ is emptied each time the rule $[\,|\,]$ is used in the derivation (this rule is the only one introducing concurrent branches). In addition, for the axiom $[\chi]$ to be used in the derivation one must have $(\_,\_) : \chi \in \Gamma$. Therefore, the only way one could have $|\mathtt{Indep}(\mathcal{G})| > 1$ (i.e., at least two concurrent branches in $\mathcal{G}$) is if χ does not appear in $\mathcal{G}$.

  Observe that the recursion is prefix guarded since we have $(\mathtt{n}_i[P_i] \mid \mathtt{n}_j[P_j])\!\updownarrow$.

- *Projection.* By induction hypothesis. □

3.5.4 Subject Reduction

In order to have a form of subject reduction for our type system, it is slightly extended to support (stable) runtime systems. Recall that a stable runtime system is such that all its queues are empty. First the rule

$$[\varepsilon]\ \frac{\Gamma \vdash S \triangleright \mathcal{G}}{\Gamma \vdash \mathtt{sr} : \varepsilon \mid S \triangleright \mathcal{G}}$$

is added so to allow empty queues to be discarded. Second, the premises of rule [0] are updated to obtain

$$[\mathbf{0}]\ \frac{\forall \mathtt{n} \in \mathcal{P}(S) :\ S(\mathtt{n}) = \mathbf{0} \qquad \mathcal{C}(S) = \varnothing}{\Gamma \vdash S \triangleright \mathbf{0}}$$

where $\mathcal{C}(S) = \varnothing$ means that there is no queue left in S, i.e., S is a program where all the participants have an empty behaviour.

The lemma below follows directly from the extension above.

Lemma 3.9. *Let S be a program,*

$$\circ \vdash S \triangleright \mathcal{G} \iff \circ \vdash S \mid \mathrm{Q}(S) \triangleright \mathcal{G}$$

Proof. ($\Rightarrow$) If $\circ \vdash S \triangleright \mathcal{G}$, then we can apply rule [ε] on $S \mid \mathrm{Q}(S)$ until all the queues have been removed and we obtain $\circ \vdash S \triangleright \mathcal{G}$ as a sub-derivation. ($\Leftarrow$) If $\circ \vdash S \mid \mathrm{Q}(S) \triangleright \mathcal{G}$, then $\circ \vdash S \triangleright \mathcal{G}$ is a sub-derivation of the runtime system. □

Theorem 3.5 (Subject reduction)**.** *Let S be a stable runtime system, if $\circ \vdash S \triangleright \mathcal{G}$ and $S \xrightarrow{\mathtt{s}\to\mathtt{r}:\mathtt{a}} \xrightarrow{\mathtt{r}\leftarrow\mathtt{s}:\mathtt{a}} S'$, then $\circ \vdash S' \triangleright \mathcal{G}'$.*

In addition, if $S \equiv \mathtt{s}[\mathtt{r}!\mathtt{a};P \oplus P'] \mid \mathtt{r}[\mathtt{s}?\mathtt{a};Q+Q'] \mid S''$*, then*

$$\mathcal{G} \equiv (\mathtt{s} \to \mathtt{r}:\mathtt{a};\ \mathcal{G}_1 + \mathcal{G}_2) \mid \mathcal{G}_3 \qquad S' \equiv \mathtt{s}[P] \mid \mathtt{r}[Q] \mid S'' \qquad \mathcal{G}' \equiv \mathcal{G}_1 \mid \mathcal{G}_3$$

Proof. Assume S is a stable runtime system such that $S \xrightarrow{\mathtt{s}\to\mathtt{r}:\mathtt{a}} \xrightarrow{\mathtt{r}\leftarrow\mathtt{s}:\mathtt{a}} S'$, from the semantics of systems and processes, it is clear that we must have

$$S \equiv \mathtt{s}\big[\mathtt{r}!\mathtt{a};P \oplus P'\big] \mid \mathtt{r}\big[\mathtt{s}?\mathtt{a};Q+Q'\big] \mid S_1$$

Since $\circ \vdash S \triangleright \mathcal{G}$, we have

$$\mathcal{G} \equiv (\mathtt{s} \to \mathtt{r}:\mathtt{a};\ \mathcal{G}_1 + \mathcal{G}_2) \mid \mathcal{G}_3$$

by Lemma 3.7 and the fact that by Theorem 3.4, $\mathcal{G}$ is well-formed and projectable.

By a similar reasoning to the one of the proof of Lemma 3.7, we have the following derivation, with $S_1 \equiv S_2 \mid S_3$ and $S_2 \not\updownarrow$.

$$[\mid]\ \frac{\nabla \qquad \dfrac{\vdots}{\circ \vdash S_3 \triangleright \mathcal{G}_3}}{\circ \vdash \mathtt{s}\big[\mathtt{r}!\mathtt{a};P \oplus P'\big] \mid \mathtt{r}\big[\mathtt{s}?\mathtt{a};Q+Q'\big] \mid S_2 \mid S_3 \triangleright \mathcal{G}} \tag{3.7}$$

where ∇ is as follows

$$[\oplus]\ \frac{[+]\ \dfrac{[;]\ \dfrac{\dfrac{\vdots}{\circ \vdash \mathtt{s}[P] \mid \mathtt{r}[Q] \mid S_2 \triangleright \mathcal{G}_1}}{\circ \vdash \mathtt{s}[\mathtt{r}!\mathtt{a};P] \mid \mathtt{r}[\mathtt{s}?\mathtt{a};Q] \mid S_2 \triangleright \mathtt{s} \to \mathtt{r}:\mathtt{a};\ \mathcal{G}_1}}{\circ \vdash \mathtt{s}[\mathtt{r}!\mathtt{a};P] \mid \mathtt{r}\big[\mathtt{s}?\mathtt{a};Q+Q'\big] \mid S_2 \triangleright \mathtt{s} \to \mathtt{r}:\mathtt{a};\ \mathcal{G}_1} \qquad \dfrac{\vdots}{\circ \vdash \mathtt{s}\big[P'\big] \mid \mathtt{r}\big[\mathtt{s}?\mathtt{a};Q+Q'\big] \mid S_2 \triangleright \mathcal{G}_2}}{\circ \vdash \mathtt{s}\big[\mathtt{r}!\mathtt{a};P \oplus P'\big] \mid \mathtt{r}\big[\mathtt{s}?\mathtt{a};Q+Q'\big] \mid S_2 \triangleright \mathtt{s} \to \mathtt{r}:\mathtt{a};\ \mathcal{G}_1 + \mathcal{G}_2}$$

From the semantics of systems, we must have

$$S' \equiv \mathtt{s}[P] \mid \mathtt{r}[Q] \mid S_1$$

and since $S_1 \equiv S_2 \mid S_3$ is not changed by reduction, we have the derivation:

$$[\mid] \dfrac{\dfrac{\vdots}{\circ \vdash \mathtt{s}[P] \mid \mathtt{r}[Q] \mid S_2 \triangleright \mathcal{G}_1} \qquad \dfrac{\vdots}{\circ \vdash S_3 \triangleright \mathcal{G}_3}}{\circ \vdash \mathtt{s}[P] \mid \mathtt{r}[Q] \mid S_2 \mid S_3 \triangleright \mathcal{G}'} \tag{3.8}$$

and the result follows from the fact that each sub-derivation of (3.8) is a sub-derivation of (3.7). □

Corollary 3.1. *Let S be a stable runtime system such that $\circ \vdash S \triangleright \mathcal{G}$. If S' is a stable runtime system such that S' is reachable from S with a 1-buffer execution, then $\circ \vdash S' \triangleright \mathcal{G}'$.*

Proof. Direct from Theorem 3.5 and the fact that closeness of behaviours is preserved by reduction. □

Lemma 3.10 below motivates the restriction of our subject reduction theorem to synchronisation of output/input. Indeed, this result shows that, for typable systems, there is always an execution for which each output is followed by its corresponding input.

Lemma 3.10. *If S is a stable runtime system such that $\circ \vdash S \triangleright \mathcal{G}$ and S' is a stable runtime system such that $S \Longrightarrow S'$, then S' is reachable by a 1-buffer execution from S.*

Proof. First we note that we have the following dependencies, by the semantics of sys-

tems. Let us define the following accessory function on labels:

$$\mathtt{subj}(\lambda) \stackrel{def}{=} \begin{cases} \mathtt{s}, & \text{if } \lambda = \mathtt{s} \to \mathtt{r} : \mathtt{a} \\ \mathtt{r}, & \text{if } \lambda = \mathtt{r} \leftarrow \mathtt{s} : \mathtt{a} \end{cases}$$

A label λ' depends on λ iff

- $\lambda = \mathtt{s} \to \mathtt{r} : \mathtt{a}$ and $\lambda' = \mathtt{r} \leftarrow \mathtt{s} : \mathtt{a}$, or
- $\mathtt{subj}(\lambda) = \mathtt{subj}(\lambda')$

We write $\lambda \prec \lambda'$ if λ' depends on λ, and we call $\prec$-causality chain a list $\lambda_1 \dots \lambda_n$ such that for all $1 \leqslant i < n$, $\lambda_i \prec \lambda_{i+1}$.

Assume, by contradiction, that we have S and S' such that $S \Longrightarrow S'$, and S' is *not* reachable by a 1-buffer execution from S. That is to say that at least two buffers are needed for S to reduce to S'. We have the following situation

$$S \xrightarrow{\mathtt{s} \to \mathtt{r} : \mathtt{a}} \xRightarrow{\phi} \xrightarrow{\mathtt{r} \leftarrow \mathtt{s} : \mathtt{a}} S'$$

where ϕ is a non-empty sequence of labels λ alternating between outputs and inputs (since S and S' are stable) and there is no 1-buffer execution (i.e., ϕ must include another label $\mathtt{s}' \to \mathtt{r}' : \mathtt{b}$). There must be a $\prec$-causality chain between the labels in ϕ and $\mathtt{s} \to \mathtt{r} : \mathtt{a}$ (resp. $\mathtt{r} \leftarrow \mathtt{s} : \mathtt{a}$), otherwise we could swap labels so that $\mathtt{r} \leftarrow \mathtt{s} : \mathtt{a}$ follows directly $\mathtt{s} \to \mathtt{r} : \mathtt{a}$. The smallest chain satisfying these requirements is $\phi = \mathtt{r} \to \mathtt{s} : \mathtt{b} \cdot \mathtt{s} \leftarrow \mathtt{r} : \mathtt{b}$. Such a sequence of transitions implies that we have the following system:

$$S \equiv \mathtt{s}[\mathtt{r}!\mathtt{a}\,;\mathtt{r}?\mathtt{b}] \mid \mathtt{r}[\mathtt{s}!\mathtt{b}\,;\mathtt{s}?\mathtt{a}] \mid S''$$

which is not typable, and thus leads to a contradiction. □

3.5.5 Equivalence with Original System

Since the branches of external choices that are discarded while typing are those that are never chosen, we can show that there is a bisimulation between the original system and the system consisting of the projections of the synthesised global type.

We first give our notion of bisimulation, then our main result (Theorem 3.6).

Definition 3.9 ($\approx$)**.** *$S \approx S'$ if and only if*

- *$S \xrightarrow{\lambda} S_1$ implies $S' \xrightarrow{\lambda} S_2$ for some $S_1 \approx S_2$, and*
- *$S' \xrightarrow{\lambda} S_2$ implies $S \xrightarrow{\lambda} S_1$ for some $S_1 \approx S_2$*

where $\lambda \in \{\mathtt{s} \to \mathtt{r} : \mathtt{a},\ \mathtt{r} \leftarrow \mathtt{s} : \mathtt{a}\}$.

Theorem 3.6. *Let S be a program, if $\circ \vdash S \triangleright \mathcal{G}$, then*

$$\Big|_{\mathtt{n} \in \mathcal{P}(S)} \mathtt{n}[\mathcal{G}\upharpoonright_{\mathtt{n}}] \ \mid\ \mathrm{Q}(S) \quad \approx \quad S \mid \mathrm{Q}(S)$$

Proof. Let $\hat{S} \equiv \Big|_{\mathtt{n} \in \mathcal{P}(S)} \mathtt{n}[\mathcal{G}\upharpoonright_{\mathtt{n}}]$. From Lemmas 3.4 and 3.8, we have that

$$S \equiv \mathtt{n}_1[P_1] \mid \ldots \mid \mathtt{n}_k[P_k]$$

such that for all $1 \leqslant \mathtt{i} \leqslant \mathtt{k}$: $\mathcal{G}\upharpoonright_{\mathtt{i}} \lesssim P_{\mathtt{i}}$. Thus, both directions of the bisimulation are straightforward if $\lambda = \mathtt{s} \to \mathtt{r} : \mathtt{a}$. We also know that each receive action done by the projected system can be simulated by the original system from Lemma 3.8.

We show that each receiving action made by the original system can also be made by the projected system, by contradiction. Let S' be such that $S \mid \mathrm{Q}(S) \Longrightarrow S'$ and $\hat{S}'$ be such that $\hat{S} \mid \mathrm{Q}(\hat{S}) \Longrightarrow \hat{S}'$; and assume that $\hat{S}'$ and S' are the first configurations that diverge (i.e., the first reduced systems where S' can do more actions that $\hat{S}'$). Let S'' (resp. $\hat{S}''$) be the

last stable system before S' (resp. $\hat{S}'$). We have

$$S \mid \mathrm{Q}(S) \Longrightarrow S'' \xrightarrow{\phi} S' \quad \text{and} \quad \hat{S} \mid \mathrm{Q}(\hat{S}) \Longrightarrow \hat{S}'' \xrightarrow{\phi} \hat{S}'$$

where ϕ consists only of sending actions. Note that ϕ must be the same in both systems since sending actions are preserved by $\lesssim$. We must have systems of the form:

$$S' \equiv \mathtt{r}\big[\mathtt{s}?\mathtt{a}\,;P + \mathtt{s}?\mathtt{b}\,;P'\big] \mid \mathtt{sr} : \mathtt{b}\cdot\rho \mid \ldots \quad \text{and} \quad \hat{S}' \equiv \mathtt{r}[\mathtt{s}?\mathtt{a}\,;P] \mid \mathtt{sr} : \mathtt{b}\cdot\rho \mid \ldots$$

where clearly $\hat{S}'$ cannot match the actions of S'. However, since S is typable and S'' is stable, we must have

$$\circ \vdash S'' \blacktriangleright \mathcal{G}''$$

by Lemma 3.10 and Theorem 3.5. Since ϕ consists only of sending actions, we must have

$$S'' \equiv \mathtt{r}\big[\mathtt{s}?\mathtt{a}\,;P + \mathtt{s}?\mathtt{b}\,;P'\big] \mid \mathtt{s}[\mathtt{r}!\mathtt{b}\ldots] \mid \ldots$$

and

$$\hat{S}'' \equiv \mathtt{r}[\mathtt{s}?\mathtt{a}\,;P] \mid \mathtt{s}[\mathtt{r}!\mathtt{b}\ldots] \mid \ldots$$

it is clear that we must have $\mathcal{G}'' \upharpoonright_{\mathtt{r}} \equiv \mathtt{s}?\mathtt{b}\,;Q + Q'$, which is in contradiction with the behaviour of $\mathtt{r}$ in $\hat{S}''$. □

3.5.6 Completeness

Our completeness result shows that every well-formed and projectable global type is inhabited by the system consisting of the parallel composition of all its projections.

We give two lemmas that are necessary for the proof of Theorem 3.7 below. Recall that the simulation relation $\lesssim$ is given in Definition 3.8 and the merge function $\_ \uplus \_$ is given in Definition 3.2.

Lemma 3.11. *Let P and Q be two processes, if $P \uplus Q \neq \bot$ then $P \lesssim P \uplus Q$.*

Proof. We show the result by induction on the structure of P and a case analysis following Definition 3.2.

- If $P \equiv Q$, the result follows trivially.
- If $P = \bigoplus_{i \in I} \mathtt{r_i}!\mathsf{a_i}; P_i$, then, by Definition 3.2, we must have $Q = \bigoplus_{i \in I} \mathtt{r_i}!\mathsf{a_i}; Q_i$, and $P_i \uplus Q_i$ for all $i \in I$. We have the result by definition of $\lesssim$ (all sending actions are preserved in both directions since P and Q have the same prefixes) and induction hypothesis: $P_i \lesssim P_i \uplus Q_i$ for all $i \in I$.
- If $P = \sum_{i \in I} \mathtt{s}?\mathsf{a_i}; P_i$, then there are two cases, either
  - $Q = \sum_{i \in I} \mathtt{s}?\mathsf{a_i}; Q_i$, then the result follows by definition of $\lesssim$ (the processes share the same prefixes) and induction hypothesis; or
  - $Q = \sum_{j \in J} \mathtt{s}?\mathsf{a_j}; Q_j$ and $\forall i \in I : \ \forall j \in J : \ \mathsf{a}_i \neq \mathsf{a}_j$ and $I, J \neq \varnothing$, then we have, by Definition 3.2,

    $$P \uplus Q = \sum_{i \in I} \mathtt{s}?\mathsf{a_i}; P_i + \sum_{j \in J} \mathtt{s}?\mathsf{a_j}; Q_j$$

    and by definition of $\lesssim$:

    $$\sum_{i \in I} \mathtt{s}?\mathsf{a_i}; P_i \ \lesssim \ \sum_{i \in I} \mathtt{s}?\mathsf{a_i}; P_i + \sum_{j \in J} \mathtt{s}?\mathsf{a_j}; Q_j$$

Note that if $P \equiv \mathbf{0}$, then we must have $Q \equiv \mathbf{0}$, cf. the condition $I, J \neq \varnothing$ above. □

Lemma 3.12. *Let S and S' be two programs such that $\mathcal{P}(S) = \mathcal{P}(S')$ and $\forall \mathtt{n} \in \mathcal{P}(S) : \ S(\mathtt{n}) \lesssim S'(\mathtt{n})$. If $\Gamma \vdash S \triangleright \mathcal{G}$ then $\Gamma \vdash S' \triangleright \mathcal{G}$.*

Proof. The proof is by straightforward induction on the derivation of $\Gamma \vdash S \triangleright \mathcal{G}$. We only give the main argument here.

First, observe that by Proposition 3.1, if $S(\mathtt{n}) \equiv \mathbf{0}$, then we must have $S'(\mathtt{n}) \equiv \mathbf{0}$.

By definition of $\lesssim$, we have that each process $S'(\mathtt{n})$ may only differ from $S(\mathtt{n})$ by having some additional external choice branches, i.e., if $S(\mathtt{n}) = P$ (with P an external choice), we can have $S'(\mathtt{n}) = P + P'$. It is easy to see that we can always apply rule [+] to discard the additional P' branch and return to the original system.

Note that, by definition of $\lesssim$, if $S(\mathtt{n}) = P \oplus P'$, then $S'(\mathtt{n}) = Q \oplus Q'$, with $P \lesssim Q$ and $P' \lesssim Q$; therefore the induction hypothesis may be applied easily for the case when the last rule is [⊕]. □

We show an intermediary result (Lemma 3.13) which allows us to take into account the set of participants of a recursive global type during its projection. We illustrate the utility of this lemma with Example 3.13 below.

Example 3.13. Consider the global type:

$$\mathcal{G} \;=\; \mu\chi.\mathtt{s} \to \mathtt{r} : \mathtt{a};\; \mathtt{s} \to \mathtt{r}' : \mathtt{b}; \chi$$

where we have $\mathcal{P}(\mathcal{G}) = \{\mathtt{s}, \mathtt{r}, \mathtt{r}'\}$. While projecting $\mathcal{G}$, we obtain the sub-term $\mathcal{G}'$:

$$\mathcal{G}' \;=\; \mathtt{s} \to \mathtt{r}' : \mathtt{b}; \chi$$

where we have $\mathcal{P}(\mathcal{G}) = \{\mathtt{s}, \mathtt{r}'\}$, i.e., $\mathtt{r}$ does not appear in $\mathcal{G}'$ anymore. However, we need to record the fact that $\mathtt{r}$ is part of the global type since $\mathtt{r}$ is involved in the recursion. ◇

Lemma 3.13. *Let* $\mathtt{Proj}(\mathcal{G}, \mathcal{Q}) = \big|_{\mathtt{n} \in \mathcal{Q}} \mathtt{n}[\mathcal{G}\upharpoonright_{\mathtt{n}}]$ *with* $\mathcal{P}(\mathcal{G}) \subseteq \mathcal{Q}$,

If $\vdash \mathcal{G}$, $\mathcal{G}$ *is projectable, and* $\forall \chi \in \mathtt{fv}(\mathcal{G}) : \exists \chi' : \forall \mathtt{n} \in \mathcal{Q} : \Gamma(\mathtt{n}, \chi') = \chi$ *then*

$$\Gamma \vdash \mathtt{Proj}(\mathcal{G}, \mathcal{Q}) \triangleright \mathcal{G}' \quad \textit{with } \mathcal{G} \equiv \mathcal{G}'$$

Proof. We show this by induction on the structure of $\mathcal{G}$. Throughout the proof, we let

$$S = \big|_{\mathtt{n}\in Q}\, \mathtt{n}[\mathcal{G}\upharpoonright \mathtt{n}] \qquad \text{and} \qquad S_{Q'} = \big|_{\mathtt{n}\in Q\setminus Q'}\, \mathtt{n}[\mathcal{G}\upharpoonright \mathtt{n}]$$

Case $\mathcal{G} = \chi$. Then, by Definition 3.1, we have

$$S \equiv \big|_{\mathtt{n}\in Q}\, \mathtt{n}[\chi]$$

By assumption, we have $\exists \chi : \ \forall \mathtt{n} \in Q : \ \Gamma(\mathtt{n}, \chi) : \chi$, hence we can apply rule [χ], and we obtain the required result.

Case $\mathcal{G} \equiv \mathbf{0}$. We have $S \equiv \big|_{\mathtt{n}\in Q}\, \mathtt{n}[\mathbf{0}]$ and the results holds by rule [0].

Case $\mathcal{G} \equiv \mathtt{s} \to \mathtt{r} : \mathtt{a}; \mathcal{G}'$.

By definition of projection, we have

$$S \equiv \mathtt{s}\big[\mathtt{r}!\mathtt{a}; \mathcal{G}'\upharpoonright_{\mathtt{s}}\big] \mid \mathtt{r}\big[\mathtt{s}?\mathtt{a}; \mathcal{G}'\upharpoonright_{\mathtt{r}}\big] \mid S_{\{\mathtt{s},\mathtt{r}\}}$$

with $Q = \mathcal{P}(S) \cup \{\mathtt{s}, \mathtt{r}\}$. We can apply rule [;] and the induction hypothesis in order to have the result, i.e.,

$$[;]\ \frac{\dfrac{}{\Gamma \vdash \mathtt{s}\big[\mathcal{G}'\upharpoonright_{\mathtt{s}}\big] \mid \mathtt{r}\big[\mathcal{G}'\upharpoonright_{\mathtt{r}}\big] \mid S_{\{\mathtt{s},\mathtt{r}\}} \blacktriangleright \mathcal{G}'}\ \text{by IH}}{\Gamma \vdash \mathtt{s}\big[\mathtt{r}!\mathtt{a}; \mathcal{G}'\upharpoonright_{\mathtt{s}}\big] \mid \mathtt{r}\big[\mathtt{s}?\mathtt{a}; \mathcal{G}'\upharpoonright_{\mathtt{r}}\big] \mid S_{\{\mathtt{s},\mathtt{r}\}} \blacktriangleright \mathtt{s} \to \mathtt{r} : \mathtt{a}; \mathcal{G}'}$$

Observe that $S_{\{\mathtt{s},\mathtt{r}\}} \not\equiv$ otherwise it would mean that

$$\exists\, \mathtt{s}' \to \mathtt{r}' : \mathtt{a} \in \mathtt{R}(\mathcal{G}') \text{ such that } \{\mathtt{s}, \mathtt{r}\} \cap \{\mathtt{s}', \mathtt{r}'\} = \varnothing$$

which is in contradiction with $\vdash \mathcal{G}$.

Case $\mathcal{G} \equiv \mathcal{G}_0 + \mathcal{G}_1$.

Let us assume that for all $\mathtt{n} \in \mathcal{P}(\mathcal{G})$ (that is not the sender on the prefixes in $\mathtt{R}(\mathcal{G})$) that $\mathcal{G}_0 \upharpoonright_\mathtt{n} \uplus \mathcal{G}_1 \upharpoonright_\mathtt{n} \neq \bot$; otherwise we can re-arrange $\mathcal{G}$ such that this condition holds since $\mathcal{G}$ is projectable.

By definition of projection, we have

$$S \equiv \mathtt{s}[\mathcal{G}_0 \upharpoonright_\mathtt{s} \oplus \mathcal{G}_1 \upharpoonright_\mathtt{s}] \mid \Big|_{\mathtt{n} \in \mathcal{Q} \setminus \{\mathtt{s}\}} \mathtt{n}[\mathcal{G}_0 \upharpoonright_\mathtt{n} \uplus \mathcal{G}_1 \upharpoonright_\mathtt{n}]$$

We can apply rule [⊕]. Indeed, by well-formedness, rule [⊕] is applicable ($\mathtt{s}$ is "preventing" the other participants to interact, similarly to the case above) and we obtain:

$$[\oplus]\ \frac{\dfrac{\vdots}{\Gamma \vdash \mathtt{s}[\mathcal{G}_0 \upharpoonright_\mathtt{s}] \mid \Big|_{\mathtt{n} \in \mathcal{Q} \setminus \{\mathtt{s}\}} \mathtt{n}[\mathcal{G}_0 \upharpoonright_\mathtt{n} \uplus \mathcal{G}_1 \upharpoonright_\mathtt{n}] \blacktriangleright \mathcal{G}_0} \qquad \dfrac{\vdots}{\Gamma \vdash \mathtt{s}[\mathcal{G}_1 \upharpoonright_\mathtt{s}] \mid \Big|_{\mathtt{n} \in \mathcal{Q} \setminus \{\mathtt{s}\}} \mathtt{n}[\mathcal{G}_0 \upharpoonright_\mathtt{n} \uplus \mathcal{G}_1 \upharpoonright_\mathtt{n}] \blacktriangleright \mathcal{G}_1}}{\Gamma \vdash S \blacktriangleright \mathcal{G}}$$

By induction hypothesis, we have the result for

$$\Gamma \vdash \mathtt{s}[\mathcal{G}_0 \upharpoonright_\mathtt{s}] \mid \Big|_{\mathtt{n} \in \mathcal{Q} \setminus \{\mathtt{s}\}} \mathtt{n}[\mathcal{G}_0 \upharpoonright_\mathtt{n}] \blacktriangleright \mathcal{G}_0 \qquad \Gamma \vdash \mathtt{s}[\mathcal{G}_1 \upharpoonright_\mathtt{s}] \mid \Big|_{\mathtt{n} \in \mathcal{Q} \setminus \{\mathtt{s}\}} \mathtt{n}[\mathcal{G}_1 \upharpoonright_\mathtt{n}] \blacktriangleright \mathcal{G}_1$$

By Lemma 3.11 and the assumption that $\mathcal{G}$ is projectable, we have

$$\forall \mathtt{n} \in \mathcal{Q} \setminus \{\mathtt{s}\} : \ \mathcal{G}_i \upharpoonright_\mathtt{n} \lesssim \mathcal{G}_0 \upharpoonright_\mathtt{n} \uplus \mathcal{G}_1 \upharpoonright_\mathtt{n} \qquad i \in \{0,1\}$$

Thus, by Lemma 3.12, we have

$$\Gamma \vdash \mathtt{s}[\mathcal{G}_i \upharpoonright_\mathtt{s}] \mid \Big|_{\mathtt{n} \in \mathcal{Q} \setminus \{\mathtt{s}\}} \mathtt{n}[\mathcal{G}_i \upharpoonright_\mathtt{n}] \blacktriangleright \mathcal{G}_i \implies \Gamma \vdash \mathtt{s}[\mathcal{G}_i \upharpoonright_\mathtt{s}] \mid \Big|_{\mathtt{n} \in \mathcal{Q} \setminus \{\mathtt{s}\}} \mathtt{n}[\mathcal{G}_0 \upharpoonright_\mathtt{n} \uplus \mathcal{G}_1 \upharpoonright_\mathtt{n}] \blacktriangleright \mathcal{G}_i$$

and we have the required result.

Case $\mathcal{G} \equiv \mathcal{G}_0 \mid \mathcal{G}_1$.

We have S of the form below, by definition of projections (and well-formedness)

$$S \equiv \mathtt{Proj}(\mathcal{G}_0, Q_0) \,|\, \mathtt{Proj}(\mathcal{G}_1, Q_1) \quad \text{with } Q_0 \cap Q_1 = \varnothing$$

Note that since $|\mathtt{Indep}(\mathcal{G})| > 1$, we have $\mathtt{fv}(\mathcal{G}) = \varnothing$, thus, by induction hypothesis, we have

$$\circ \vdash \mathtt{Proj}(\mathcal{G}_i, Q_i) \triangleright \mathcal{G}_i' \quad \text{and} \quad \mathcal{G}_i \equiv \mathcal{G}_i'$$

for $i \in \{0,1\}$ and we have the result by applying rule $[\,|\,]$:

$$[\,|\,]\ \frac{\dfrac{}{\circ \vdash \mathtt{Proj}(\mathcal{G}_0, Q_0) \triangleright \mathcal{G}_1}\ \text{by IH} \qquad \dfrac{}{\circ \vdash \mathtt{Proj}(\mathcal{G}_1, Q_1) \triangleright \mathcal{G}_1}\ \text{by IH}}{\Gamma \vdash S \triangleright \mathcal{G}}$$

Case $\mathcal{G} \equiv \mu\chi.\mathcal{G}'$.

By definition of projections, we have

$$S \equiv \big|_{\mathtt{n} \in Q_1} \mathtt{n}\big[\mu\chi.\mathcal{G}' \upharpoonright \mathtt{n}\big] \;\big|\; \big|_{\mathtt{n} \in Q_2} \mathtt{n}[\mathbf{0}] \qquad \text{with } Q = Q_1 \cup Q_2 \text{ and } Q_1 \cap Q_2 = \varnothing$$

Since $\mathcal{G}'$ is prefix-guarded, there must be $\mathtt{s}, \mathtt{r} \in Q_1$ such that

$$\mathtt{s}\big[\mu\chi.\mathcal{G}' \upharpoonright \mathtt{s}\big] \,|\, \mathtt{r}\big[\mu\chi.\mathcal{G}' \upharpoonright \mathtt{r}\big]\!\updownarrow$$

Therefore, rule $[\mu]$ is applicable after having discarded the empty processes via rule $[\,|\,]$

(followed by the axiom [0]). We have the following derivation:

$$\dfrac{\dfrac{\dfrac{\text{by IH}}{\Gamma' \vdash \big|_{\mathtt{n}\in Q_1}\, \mathtt{n}\big[\mathcal{G}'\upharpoonright_{\mathtt{n}}\big] \triangleright \mathcal{G}'}\,[\mu] \qquad \dfrac{}{\Gamma \vdash \big|_{\mathtt{n}\in Q_2}\, \mathtt{n}[\mathbf{0}] \triangleright \mathbf{0}}\,[\mathbf{0}]}{\Gamma \vdash \big|_{\mathtt{n}\in Q_1}\, \mathtt{n}\big[\mu\chi.\mathcal{G}'\upharpoonright_{\mathtt{n}}\big] \triangleright \mathcal{G} \quad\big|\quad \big|_{\mathtt{n}\in Q_2}\, \mathtt{n}[\mathbf{0}] \triangleright \mathcal{G} \,|\, \mathbf{0}}\,[\,|\,]$$

$$\text{where} \qquad \Gamma' = \Gamma \cdot \Gamma_S \quad \text{and} \quad \Gamma_S \text{ such that } \forall\, \mathtt{n} \in Q_1 : \Gamma(\mathtt{n},\chi) = \chi$$

We have the required result by induction hypothesis. □

Theorem 3.7. *If $\vdash \mathcal{G}$ and $\mathcal{G}$ is projectable, then there is $\mathcal{G}' \equiv \mathcal{G}$ such that*

$$\circ \vdash \big|_{\mathtt{n}\in \mathcal{P}(\mathcal{G})}\, \mathtt{n}\big[\mathcal{G}\upharpoonright_{\mathtt{n}}\big] \triangleright \mathcal{G}'$$

Proof. By Lemma 3.13, with $Q = \mathcal{P}(\mathcal{G})$ and $\Gamma = \circ$ (since $\mathcal{G}$ is closed by assumption). □

3.6 Perspectives

In this section, we discuss possible generalisations of our framework to extend the set of typable systems and discuss a limitation of the synthesis as a type system.

Concurrent branches and recursion. In order to synthesise global types where concurrent branches appear under recursion (i.e., a construct which is not allowed in the current setting), we propose to introduce an additional recursion variable environment which is used linearly. We use judgements of the form

$$\hat{\Gamma}; \Gamma \vdash S \triangleright \mathcal{G}$$

which may be read as "the system S forms a choreography defined by a global type $\mathcal{G}$, under the environments $\hat{\Gamma}$ and Γ". Environments $\hat{\Gamma}$ and Γ are as in Section 3.4, except from the fact that each of the "hypothesis" in $\hat{\Gamma}$ must be used exactly once. The rules from Section 3.4.1 remains essentially the same but for the rule introducing parallel branches and the axioms for termination and recursion variable. We give the new rules below.

$$[\,|\,]\ \frac{\hat{\Gamma}\cdot\Gamma;\circ\vdash S\triangleright\mathcal{G}\qquad\hat{\Gamma}\cdot\Gamma';\circ\vdash S'\triangleright\mathcal{G}'}{\hat{\Gamma};\Gamma\cdot\Gamma'\vdash S\,|\,S'\triangleright\mathcal{G}\,|\,\mathcal{G}'}$$

$$[\chi]\ \frac{\forall 1\leqslant \mathtt{i}\leqslant k:\ \Gamma(\mathtt{n}_\mathtt{i},\chi_\mathtt{i})=\chi}{\circ;\Gamma\vdash \mathtt{n}_1[\chi_1]\,|\,\ldots\,|\,\mathtt{n}_\mathtt{k}[\chi_\mathtt{k}]\triangleright\chi}\qquad [\mathbf{0}]\ \frac{\forall \mathtt{n}\in\mathcal{P}(S):\ S(\mathtt{n})=\mathbf{0}}{\circ;\Gamma\vdash S\triangleright\mathbf{0}}$$

$$[\chi_L]\ \frac{\mathtt{dom}(\hat{\Gamma})=\{(\mathtt{n}_\mathtt{i},\chi_\mathtt{i})\mid 1\leqslant\mathtt{i}\leqslant\mathtt{k}\}\qquad \mathtt{img}(\hat{\Gamma})=\{\chi\}}{\hat{\Gamma};\Gamma\vdash \mathtt{n}_1[\chi_1]\,|\,\ldots\,|\,\mathtt{n}_\mathtt{k}[\chi_\mathtt{k}]\triangleright\chi}$$

Rule $[\,|\,]$ allows to separate the system in two concurrent branches. In this case the normal environment Γ is split in two disjoint parts and moved to the linear environments. Rules $[\chi]$ and $[\mathbf{0}]$ are updated so that they are applicable only when the linear environment is empty. The new rule $[\chi_L]$ allows to use the variables stored in $\hat{\Gamma}$ only if each behaviour in the system consists of recursion variables in $\hat{\Gamma}$ and each element in $\hat{\Gamma}$ is indeed used. Intuitively, these rules allows some participants to agree on a "meeting point" (i.e., a recursion definition), then they behave concurrently for a while, and return to the meeting point eventually.

Another change in rules concerns rule $[\mu]$ where we add the constraint that there must be exactly one pair of participants that interact after the recursion. This is to prevent rule

$[\mu]$ to be applicable on systems of the form

$$\mathtt{s}_1[\mu\chi.\mathtt{r}_1!\mathtt{a};\chi] \mid \mathtt{r}_1[\mu\chi.\mathtt{s}_1?\mathtt{a};\chi] \mid \mathtt{s}_2[\mu\chi.\mathtt{r}_2!\mathtt{b};\chi] \mid \mathtt{r}_2[\mu\chi.\mathtt{s}_2?\mathtt{b};\chi]$$

where it is clear that rule $[\,|\,]$ should be used before rule $[\mu]$ so that we obtain a global type of the form:

$$\mu\chi.\mathtt{s}_1 \to \mathtt{r}_1 : \mathtt{a}; \chi \mid \mu\chi'.\mathtt{s}_2 \to \mathtt{r}_2 : \mathtt{b}; \chi' \qquad \checkmark$$

Instead of a global type of the form:

$$\mu\chi.(\mathtt{s}_1 \to \mathtt{r}_1 : \mathtt{a}; \chi \mid \mathtt{s}_2 \to \mathtt{r}_2 : \mathtt{b}; \chi) \qquad \times$$

We show an example of a system that would be rejected by the original rules but is accepted by this extension.

Example 3.14. Consider the system

$$S = \mathtt{s}\big[\mu\chi.\mathtt{r}!\mathtt{a};\mathtt{r}'!\mathtt{b};\chi\big] \mid \mathtt{r}\big[\mu\chi.\mathtt{s}?\mathtt{a};\mathtt{s}'?\mathtt{c};\chi\big] \mid \mathtt{r}'[\mu\chi.\mathtt{s}?\mathtt{b};\chi] \mid \mathtt{s}'[\mu\chi.\mathtt{r}!\mathtt{c};\chi]$$

which has the following global type

$$\mathcal{G} = \mu\chi.\mathtt{s} \to \mathtt{r} : \mathtt{a}; (\mathtt{s} \to \mathtt{r}' : \mathtt{b}; \chi \mid \mathtt{s}' \to \mathtt{r} : \mathtt{c}; \chi)$$

We give the type derivation below, where we pose

$$\Gamma_1 = (\mathtt{s}, \chi) : \chi \cdot (\mathtt{r}', \chi) : \chi \quad \text{and} \quad \Gamma_2 = (\mathtt{r}, \chi) : \chi \cdot (\mathtt{s}', \chi) : \chi$$

and we abstract from the intermediary global types in the derivation.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[\chi_L]}{\Gamma_1\,;\circ \vdash \mathtt{s}[\chi]\ |\mathtt{r}'[\chi] \triangleright \chi}[;]}{\Gamma_1\,;\circ \vdash \mathtt{s}\big[\mathtt{r}'!\mathtt{b};\chi\big]\ |\mathtt{r}'[\mathtt{s}?\mathtt{b};\chi] \triangleright \ldots} \qquad \dfrac{\dfrac{[\chi_L]}{\Gamma_2\,;\circ \vdash\ |\mathtt{r}[\chi]\ |\mathtt{s}'[\chi] \triangleright \chi}[;]}{\Gamma_2\,;\circ \vdash\ |\mathtt{r}\big[\mathtt{s}'?\mathtt{c};\chi\big]\ |\mathtt{s}'[\mathtt{r}!\mathtt{c};\chi] \triangleright \ldots}}{\circ\,;\Gamma_1\cdot\Gamma_2 \vdash \mathtt{s}\big[\mathtt{r}'!\mathtt{b};\chi\big]\ |\mathtt{r}\big[\mathtt{s}'?\mathtt{c};\chi\big]\ |\mathtt{r}'[\mathtt{s}?\mathtt{b};\chi]\ |\mathtt{s}'[\mathtt{r}!\mathtt{c};\chi] \triangleright \ldots}[\,|\,]}{\circ\,;\Gamma_1\cdot\Gamma_2 \vdash \mathtt{s}\big[\mathtt{r}!\mathtt{a};\mathtt{r}'!\mathtt{b};\chi\big]\ |\mathtt{r}\big[\mathtt{s}?\mathtt{a};\mathtt{s}'?\mathtt{c};\chi\big]\ |\mathtt{r}'[\mathtt{s}?\mathtt{b};\chi]\ |\mathtt{s}'[\mathtt{r}!\mathtt{c};\chi] \triangleright \ldots}[;]}{\circ\,;\circ \vdash S \triangleright \mathcal{G}}[\mu]$$

Notice that, when rule $[\,|\,]$ is used, the environment $\Gamma_1 \cdot \Gamma_2$ is split in two parts and moved to the linear environment. $\diamond$

Remarkably, if we relax the corresponding conditions on the well-formedness of global types and their projections, the properties presented in Section 3.5 also holds for this generalisation. In fact, the main change concerns *knowledge of choice*. This property holds in this extension since each participant involved in a "concurrent recursion" must *always* return to the recursion point.

Minimal local types. Some system of local types may fail to be assigned a global type because the behaviours of two participants do not match within a recursion, even if they do not deadlock – this is due to the syntactic nature of the type system. We may be given more chances to type a system of local types if each local type has been *minimised* up to bisimilarity beforehand. For instance, the system:

$$\mathtt{s}[\mu\chi.\mathtt{r}!\mathtt{a};\chi]\ |\ \mathtt{r}[\mu\chi.\mathtt{s}?\mathtt{a};\mathtt{s}?\mathtt{a};\chi] \tag{3.9}$$

is not typable because the typing derivation will eventually reach a term of the form

$$(\mathtt{s},\chi):\chi\cdot(\mathtt{r},\chi):\chi \vdash \mathtt{s}[\chi] \mid \mathtt{r}[\mathtt{s?a};\chi] \blacktriangleright \ldots$$

for which no rule from Figure 3.3 is applicable. In particular, observe that unfolding any of the behaviours in (3.9) will not help.

However, the system in (3.9) is typable if we minimise each behaviour. Indeed, it is possible to translate the behaviour of $\mathtt{r}$ into an automaton, which can be minimised and translated back to the process: $\mu\chi.\mathtt{s?a};\chi$. Note that, at the level of behavioural specifications, minimising a process (up to bisimilarity) is a sound transformation.

After minimisation, the system in (3.9) is assigned the global type $\mu\chi.\mathtt{s}\to\mathtt{r}:\mathtt{a};\chi$. It is easy to see that the system consisting of the projections of this global type is bisimilar to the original one, i.e.,

$$\mathtt{s}[\mu\chi.\mathtt{r!a};\chi] \mid \mathtt{r}[\mu\chi.\mathtt{s?a};\mathtt{s?a};\chi] \quad\approx\quad \mathtt{s}[\mu\chi.\mathtt{r!a};\chi] \mid \mathtt{r}[\mu\chi.\mathtt{s?a};\chi]$$

The result follows directly from the fact that minimisation is done up to bisimilarity and the fact that a synthesised global type preserve the behaviour of the original system (cf. Theorem 3.6).

A limitation. A disadvantage of presenting a synthesis as a type system is that both checking that each input/output actions are matched and checking that there is a 1-buffer execution of the system (between two stable configurations) must be done at the same time. Instead, when working with communicating machines, one can first construct a transition system corresponding to all the possible (bounded) executions of the systems, so to check that, e.g., the system is deadlock free, there is no orphan message configuration etc. Then, one may check that every stable configuration is reachable by a 1-buffer execution and, therefore, can be captured by a global type.

This limitation is most noticeable if we were to allow a more liberal version of the external choice in the behaviours, e.g., a behaviour of the form:

$$P = \mathtt{s}?\mathtt{a}\,;\mathtt{s}'?\mathtt{b} \;+\; \mathtt{s}'?\mathtt{b}\,;\mathtt{s}?\mathtt{c} \qquad \text{with } \mathtt{s} \neq \mathtt{s}'$$

Indeed, consider the following system

$$S = \mathtt{s}\big[\mathtt{r}!\mathtt{a}\,;\mathtt{s}'!\mathtt{d}\big] \;\mid \mathtt{r}[P] \;\mid \mathtt{s}'[\mathtt{s}?\mathtt{d}\,;\mathtt{r}!\mathtt{b}]$$

In a 1-buffer execution, this system does not deadlock. Indeed, under such a restriction $\mathtt{r}$ must read $\mathtt{a}$ from the queue $\mathtt{sr}$ first, then $\mathtt{b}$ from the queue $\mathtt{s}'\mathtt{r}$. However, the runtime system $S \mid \mathrm{Q}(S)$ may reduce to the following system:

$$S' = \mathtt{s}[\mathbf{0}] \;\mid \mathtt{sr} : \mathtt{a} \;\mid \mathtt{r}\big[\mathtt{s}?\mathtt{a}\,;\mathtt{s}'?\mathtt{b} \;+\; \mathtt{s}'?\mathtt{b}\,;\mathtt{s}?\mathtt{c}\big] \;\mid \mathtt{s}'\mathtt{r} : \mathtt{b} \;\mid \mathtt{s}'[\mathbf{0}]$$

in which case if $\mathtt{r}$ reads $\mathtt{b}$ first (i.e., the right branch of the external choice is chosen), it will deadlock because it will not be able to read $\mathtt{c}$ from the queue $\mathtt{sr}$.

With the current type system, it is not possible to discard such systems as we must eventually discard one of the branches of participant $\mathtt{r}$ and thus the typing will not check the "faulty" branch. Indeed, this branch is "invisible" to the type system which only considers 1-buffer executions.

3.7 Concluding Remarks

We presented a type system to synthesise a choreography (i.e., a global type) from a set of local specifications (i.e., local types). Such a global type is unique, well-formed, and its projections are equivalent to the original local types. We have shown safety and progress properties for (typable) local session types and given a subject reduction theorem for our

type system.

Observe that our type system ensures that a synthesised global type meets the connectedness conditions of [50]. This follows from the fact that a synthesised global type is well-formed: local choice and sequentiality hold.

In addition, considering a set of typable local types as an implementation of its synthesised global type, the properties identified in [29], i.e., sequentiality, alternativeness, shuffling, fitness and exhaustivity, are guaranteed to hold.[5] This follows from the fact that a global type is well-formed and its projections are equivalent to the original system, i.e., its "implementation". In particular, *exhaustivity* – requiring that if a sequence of interactions is specified by a global type, then there must exist at least an execution of its implementation that exhibits these interactions – is guaranteed by construction.

In the rest of this thesis, we will present some applications of the synthesis. In Chapter 4, we show that once a choreography is available, it may be decorated with logic predicates so to constrain interactions declaring senders obligations and receivers requirements on the values of the exchanged data and on the choice of the branches to follow. In particular, we focus on algorithms and a methodology to correct these choreographies if they do not satisfy some requirements. In Chapter 5, we present a formal framework for distributed systems where the synthesis of choreographies is used as basis for finding sets of compliant contracts.

[5] See Section 2.2.1 for a description of these properties.

CHAPTER 4

Amending Contracts for Choreographies

We show a few techniques that help software architects to amend *global assertions* during the design of distributed choreographies. Our results include: two algorithms to solve history sensitivity problems, one algorithm to solve temporal satisfiability problems, and a methodology for applying the algorithms to protocol design.

4.1 Introduction

Once a global type has been designed, or synthesised as in Chapter 3, it may be refined into a *global assertion* [12]. That is, a global type decorated with predicates that constrain interactions, declaring senders obligations and receivers requirements on exchanged data and on the choice of the branches to follow.

Once designed, a global assertion $\mathcal{G}$ may be projected on endpoint assertions that are local types – modelling the behaviour of a specific participant – constrained according to the predicates of $\mathcal{G}$. However, a global assertion may be projected into "safe" local types only if it is *well-asserted* [12], namely when it obeys two precise design principles: *history-sensitivity* (HS for short) and *temporal satisfiability* (TS for short). Informally, HS demands that a participant having an obligation on a predicate has enough information for

choosing a set of values that guarantees it. Instead, TS requires that the values sent in each interaction do not make predicates of future interactions unsatisfiable.

The main motivation of our interest in HS and TS is that, in global assertions, they are the technical counterparts of the fundamental coordination issue that could be summarized in the slogan "who do what and when do they do it?". In fact, HS pertains to *when* variables are constrained and *who* constrain them, while TS pertains to *which* values variables take. The contracts specified in global assertions are, on the one hand, "global" as they pertain to the whole choreography while, on the other hand, they are also "local" in (at least) two aspects. The first is that they assign responsibilities to participants (*who*) at definite moments of the computation (*when*). The second aspect is that the values assigned to variables are critical because one could either over-constrain variables fixed in the past or over-restrict the range of those assigned in the future (*which*). These conditions – especially TS – are rather crucial as global assertions that violate them may be infeasible or fallacious. Remarkably, a global assertion not satisfying TS may lead to conversations in which progress is not guaranteed unless one of the participants deliberately violates the contract.

Guaranteeing HS and TS is often non-trivial, and this burden is on the software architect; using tools like the ones described in [52], one only highlights the problems but does not help to fix them. HS and TS are global semantic properties that may be hard to achieve. Namely, TS requires to trace back for "under-constrained" interactions (i.e., which allow values causing future predicates to be unsatisfiable) and re-distribute there the unsatisfiable constraints.

A motivating example. Using the running example of Chapter 3, we illustrate how TS problems may easily occur at design time. In the global type depicted in Figure 4.1, a buyer $\texttt{b}_\texttt{i}$ requests a quote from a seller $\texttt{s}_\texttt{i}$, then depending on whether the client is satisfied with the quote, $\texttt{b}_\texttt{i}$ chooses either the ok branch and the interactions terminate, or

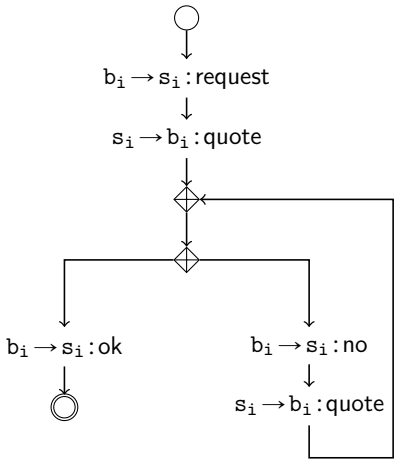


Figure 4.1: Seller-Buyer global type (extract)

$\mathtt{b_i}$ chooses the no branch and the seller has to offer a new quote.

A natural refinement of this global type is the following global assertion:[1]

$$
\begin{aligned}
\hat{\mathcal{G}} \quad = \quad & \mathtt{b_i} \to \mathtt{s_i} : \{ \mathit{request} \mid \mathbf{true} \}; && (1)\\
& \mathtt{s_i} \to \mathtt{b_i} : \{ \mathit{quote}_f \mid \mathbf{true} \}; && (2)\\
& \mu\chi\langle \mathit{quote}_f\ \mathit{quote}_f \rangle \{ \mathit{quote}_c\ \mathit{quote}_p \mid \mathit{quote}_c \leqslant \mathit{quote}_p \}. && (3)\\
& \mathtt{b_i} \to \mathtt{s_i} : \ \{ \quad \{\mathbf{true}\}\ \mathsf{ok} : \ \mathbf{0}, && (4)\\
& \qquad\qquad \{\mathbf{true}\}\ \mathsf{no} : \ \mathtt{s_i} \to \mathtt{b_i} : \{ \mathit{quote}_n \mid \mathit{quote}_n < \mathit{quote}_c \}; && (5)\\
& \qquad\qquad\qquad\qquad \chi\langle \mathit{quote}_n\ \mathit{quote}_c \rangle && (6)\\
& \qquad \}
\end{aligned}
$$

where *request*, quote_f, quote_c, quote_p and quote_n are *interaction variables*; we assume that the sort of *request* is string and the sort of the other variables is int – for simplicity.

In lines (1-2), the buyer and the seller exchange variables *request* and quote_f. The variables are unconstrained (the predicates are set to true).

Line (3) defines a recursion over the recursion parameters quote_c and quote_p – which

[1] Observe that the structure of $\hat{\mathcal{G}}$ is the same as the global type of Figure 4.1.

are both initialised to the value of $quote_f$. The invariant of the recursion, i.e., $quote_c \leqslant quote_p$ requires that the current quote ($quote_c$) is less than the previous one ($quote_p$). This invariant trivially holds in the first iteration.

At line (4), $\mathtt{b_i}$ selects either the ok or no branch (the choice of a branch may be constrained by a predicate, but in this case they are both set to **true**, so to model a non-deterministic choice).

In the recursive branch, at lines (5-6), the seller has to send a new quote ($quote_n$) to the buyer and this quote must be (strictly) less than the previous quote, i.e., $quote_n < quote_c$. The recursive call $\chi\langle quote_n\ quote_c\rangle$ says that the recursion arguments are now $quote_n$ and $quote_c$ (i.e., the current quote and the previous quote).

We will see that $\hat{\mathcal{G}}$ is *well-asserted*. However, the designer may soon realise that $\hat{\mathcal{G}}$ is under-constrained as it may imply that the seller has to make offers that are arbitrarily low. Assume that the designer tackles this flaw by updating the recursion invariant (line (3)) to the predicate

$$\mathtt{MIN} < quote_c \leqslant quote_p \tag{4.1}$$

where MIN is a constant representing the minimum price that a seller is willing to sell an item. In fact, this change makes the global assertion violate temporal satisfiability since, e.g., if $\mathtt{s_i}$ sends a quote (at line (2) or (5)) that is less than MIN, then the invariant (4.1) does not hold.

In the rest of this chapter, we will see that solving this kind of problems is not trivial, but may be solved automatically by our algorithms and methodology.

4.2 Preliminaries

We now define global assertions formally. Let $\mathcal{V}$ (ranged over by $u, v, x, y, \ldots$) be a countably infinite set of *interaction variables*, such that $\mathbb{P} \cap \mathcal{V} = \varnothing$. Recall that $\mathbb{P}$ is the set of

participants of a choreography (cf. Chapter 3).

Hereafter, $\vec{\_}$ represents a list of some elements (for instance, $\vec{v}$ is a list of interaction variables). The concatenation of $\vec{x}$ and $\vec{y}$ is denoted by the juxtaposition $\vec{x}\,\vec{y}$; abusing notation, we identify a one-element list with its (unique) element and identify lists with the underlying sets of their elements (e.g., $a \in \vec{x}$ indicates that a occurs in the list $\vec{x}$).

As in [12], we parametrise our constructions by abstracting away from the logical language Ψ adopted. Here, it suffices to assume that Ψ is a decidable fragment of a first-order logic obtained by adding first-order quantification to a language of boolean expressions. In fact, we allow expressions (ranged over by e) that include constructors and operators/relations of common data types (e.g., strings, integers, booleans, etc.) and include variables drawn from $\mathcal{V}$. (For simplicity, our examples use basic numeric types or strings.) We write $\mathtt{var}(e)$ to denote the set of variables occurring in e and use the symbol $\Longrightarrow$ to denote logic implication. Then a predicate $\psi \in \Psi$ is either a boolean expression e (understood to be a boolean expression in our language of expressions), or a quantified predicate $\forall\,\vec{v}:\ e$ or $\exists\,\vec{v}:\ e$. Given a predicate ψ in Ψ, $\mathtt{var}(\psi)$ is the set of free interaction variables of ψ (we write $\psi(\vec{v})$ to emphasise that $\mathtt{var}(\psi) \subseteq \vec{v}$).

The main ingredients of global assertions are *interactions*, abbreviated ι, like

$$\mathtt{s} \to \mathtt{r} : \{\vec{v} \mid \psi\} \tag{4.2}$$

where $\mathtt{s}, \mathtt{r} \in \mathbb{P}$ are the *sender* and the *receiver*, $\vec{v} \subseteq \mathcal{V}$ is a pairwise distinct list of variables, and $\psi \in \Psi$. We say that the variables $\vec{v}$ in (4.2) are *introduced* by $\mathtt{s}$. The interaction (4.2) reads as "$\mathtt{s}$ has to send to $\mathtt{r}$ some values for $\vec{v}$ that satisfy ψ" or as "$\mathtt{r}$ relies on the fact that the values fixed by $\mathtt{s}$ for $\vec{v}$ satisfy ψ". For instance,[2]

$$\mathtt{s} \to \mathtt{r} : \{v\,w \mid \exists u:\ v = u \times w\}$$

[2]For simplicity, we assume the typing of variables understood.

states that $\mathtt{s}$ has the obligation to send $\mathtt{r}$ two values such that the first is a multiple of the second. Given ι as in (4.2), we define

$$\mathtt{snd}(\iota) \stackrel{def}{=} \mathtt{s}, \qquad \mathtt{rcv}(\iota) \stackrel{def}{=} \mathtt{r}, \qquad \mathtt{var}(\iota) \stackrel{def}{=} \vec{v}, \qquad \text{and} \qquad \mathtt{cst}(\iota) \stackrel{def}{=} \psi$$

Remark 4.1. *In [12], interactions specify a channel over which participants communicate. For consistency with the rest of this thesis, we omit channels since they are inconsequential to our results. In fact, the algorithms we present do not use identities of channels but only those of participants and variables.*

Global assertions are ranged over by $\mathcal{G}$ and have the following syntax:

$$\begin{array}{rcll}
\mathcal{G} & ::= & \iota;\, \mathcal{G} & \text{[PREFIX]} \\
& \mid & \mathtt{s} \to \mathtt{r} : \left\{ \{\psi_j\} \mathtt{l}_j : \mathcal{G}_j \right\}_{j \in J} & \text{[BRANCHING]} \\
& \mid & \mu\chi\langle \vec{e} \rangle \{ \vec{v} \mid \psi \}.\, \mathcal{G} & \text{[RECURSIVE DEFINITION]} \\
& \mid & \chi\langle \vec{e} \rangle & \text{[RECURSIVE CALL]} \\
& \mid & \mathbf{0} & \text{[END SESSION]}
\end{array}$$

where $\psi, \psi_j \in \Psi$ and $\mathtt{l}_j$ ranges over a set of labels.

The syntax above is essentially borrowed from [12] but for a slightly simplified notation. In [12], the semantics of global assertions is given in terms of *endpoint assertions* (by projecting global assertions to endpoint assertions and exploiting the operational semantics of the latter). In this chapter, only the syntactic aspects of global assertions given below are relevant; therefore, we give an informal account of the semantics of global assertions.

The prefix production $\iota;\, \mathcal{G}$ defines a global assertion where the interaction ι must precede the interactions in $\mathcal{G}$. The branching production allows the selector $\mathtt{s}$ to choose one of the labels $\mathtt{l}_j$ and send it to $\mathtt{r}$, then the interactions in $\mathcal{G}_j$ occur. Recursion is dealt with

as usual but for the presence of an initialisation vector $\vec{e}$ (of the same length as $\vec{v}$) which specifies the initial values of each formal parameter in $\vec{v}$ and onto which a recursion invariant ψ is specified. Finally, the last production represents a completed global assertion; trailing occurrences of $\mathbf{0}$ are often omitted.

In a recursive definition $\mu\chi\langle\vec{e}\rangle\{\vec{v} \mid \psi\}.\,\mathcal{G}$, occurrences of χ (i.e., recursive calls) in $\mathcal{G}$ must be prefix-guarded and the length of $\vec{e}$ is the same as $\vec{v}$. Also, we assume that variables χ are always in the scope of a recursive definition $\mu\chi\langle\_\rangle\{\_ \mid \_\}$, and that, in a recursive call $\chi\langle\vec{e}\rangle$, $\vec{e}$ match the sorts of its corresponding recursive definition $\mu\chi\langle\vec{e}'\rangle\{\vec{v} \mid \psi\}.\,\mathcal{G}$ and the length of $\vec{e}$ is the same as $\vec{v}$.

We denote with $\mathtt{var}(\mathcal{G})$ the set of interaction variables and recursion parameters in $\mathcal{G}$. The interaction variables $\mathtt{var}(\iota)$ of global assertion $\iota;\,\mathcal{G}$ are bound in $\mathcal{G}$ and in $\mathtt{cst}(\iota)$; similarly, the formal parameters $\vec{v}$ in a recursive definition $\mu\chi\langle\_\rangle\{\vec{v} \mid \psi\}.\,\mathcal{G}$ are bound in ψ and in the recursion body $\mathcal{G}$. We consider *closed* global assertions (i.e., for any occurrence of $v \in \mathcal{V}$ in $\mathcal{G}$ either the occurrence is in a recursive definition having v as formal parameter or there is an interaction ι in $\mathcal{G}$ such that $v \in \mathtt{var}(\iota)$ which precedes that occurrence of v).

Remark 4.2. *For simplicity, we assume that bound interaction variables are pairwise distinct.*

Remark 4.3. *For consistency with [12], we use a restricted form of branching, where a choice must be made between exactly two participants (one sender and one receiver). However, we conjecture that our results are applicable to global assertions that features constructs of the form*

$$\sum_{\mathtt{j}\in J} \mathtt{s} \to \mathtt{r_j} : \{\mathsf{l_j} \mid \psi_j\};\, \mathcal{G}_j$$

where we may have $\mathtt{r_j} \neq \mathtt{r_i}$ *for* $\mathtt{i} \neq \mathtt{j} \in J$.

Definition 4.1 (Knows)**.** *Under the syntactic restrictions listed above, we say that a participant* p knows *a variable* $v \in \mathtt{var}(\mathcal{G})$ *if one of the following conditions holds:*

- *there is* ι *in* $\mathcal{G}$ *such that* $v \in \mathtt{var}(\iota)$ *and* $\mathtt{p} \in \{\mathtt{snd}(\iota), \mathtt{rcv}(\iota)\}$ *or*
- *there is a recursive definition* $\mu\chi\langle \vec{e}_1\ e\ \vec{e}_2\rangle\{\vec{v}_1\ v\ \vec{v}_2 \mid \psi\}.\,\mathcal{G}'$ *in* $\mathcal{G}$ *such that* $\mathtt{p}$ *knows all the variables in* $\mathtt{var}(e)$ *and, for each recursive invocation* $\chi\langle \vec{e'}_1\ e'\ \vec{e'}_2\rangle$ *in* $\mathcal{G}'$, $\mathtt{p}$ *knows all the variables in* $\mathtt{var}(e')$.

We denote with $\mathtt{knows}_{\mathtt{p}}(\mathcal{G})$ *the set of variables in* $\mathtt{var}(\mathcal{G})$ *that* $\mathtt{p}$ *knows.*

Example 4.1. Consider the following global assertion

$$
\begin{aligned}
\mathcal{G}_{\text{ex4.1}} \;=\;& \mathtt{I} \to \mathtt{Server}\colon \{x \mid x \geqslant 3\};\\
& \mu\chi\langle 3\rangle\{r \mid \mathbf{true}\}.\,\mathtt{Server} \to \mathtt{Player}\colon \{\\
& \quad \{r > x\}\ \mathsf{less} : \mathtt{Player} \to \mathtt{Server}\colon \{y \mid \mathbf{true}\};\ \chi\langle y\rangle,\\
& \quad \{r < x\}\ \mathsf{greater} : \mathtt{Player} \to \mathtt{Server}\colon \{z \mid \mathbf{true}\};\ \chi\langle z\rangle,\\
& \quad \{r = x\}\ \mathsf{win} : \mathbf{0}\ \}
\end{aligned}
$$

where $\mathtt{I}$ initialises a value $x \geqslant 3$ for $\mathtt{Server}$. Then, repeatedly, $\mathtt{Server}$ sends a label chosen in the set $\{\mathsf{less}, \mathsf{greater}, \mathsf{win}\}$ to $\mathtt{Player}$ depending on r being greater, smaller, or equal to the value of x; and $\mathtt{Player}$ replies with an integer in the first two cases while the interaction ends if win was sent by $\mathtt{Server}$. In $\mathcal{G}_{\text{ex4.1}}$, both $\mathtt{I}$ and $\mathtt{Server}$ know x while $\mathtt{Player}$ does not know it; instead the recursion parameter r is known only to $\mathtt{Server}$ and $\mathtt{Player}$. ◇

It is convenient to treat global assertions as trees whose nodes are drawn from a set $\mathcal{N}$ (ranged over by $n, n', \ldots$) and labelled with information on the syntactic categories of the syntax of global assertions. Hereafter, we write $n \in T$ if n is a node of a tree T, $\underline{n}$ to denote the label of n, and $T^\bullet$ for the root of T.

Definition 4.2 (Assertion Tree)**.** *The* assertion tree $\mathtt{T}(\mathcal{G})$ *of a global assertion* $\mathcal{G}$ *is defined as follows:*

- *If* $\mathcal{G} = \iota;\ \mathcal{G}'$ *then* $\mathtt{T}(\mathcal{G})^\bullet$ *has label* ι *and its unique child is* $\mathtt{T}(\mathcal{G}')$.

- *If* $\mathcal{G} = \mathtt{s} \to \mathtt{r} : \{\{\psi_j\}\mathsf{l}_j : \mathcal{G}_j\}_{j\in J}$ *then* $\mathtt{T}(\mathcal{G})^\bullet$ *has label* $\mathtt{s} \to \mathtt{r}:$ *and its children are* $\{n_j\}_{j\in J} \subseteq \mathcal{N}$ *such that, for each* $j \in J$, $\underline{n_j} = \{\psi_j\}\mathsf{l}_j$ *and* $\mathtt{T}(\mathcal{G}_j)$ *is the unique child of* n_j.

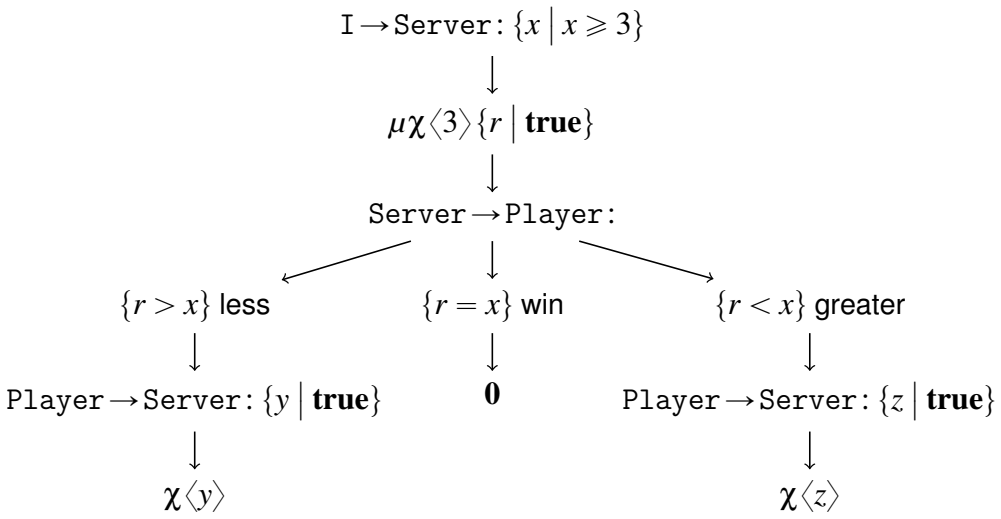
- *If* $\mathcal{G} = \mu\chi\langle\vec{e}\rangle\{\vec{v} \mid \psi\}.\,\mathcal{G}'$ *then* $\mathtt{T}(\mathcal{G})^\bullet$ *has label* $\mu\chi\langle\vec{e}\rangle\{\vec{v} \mid \psi\}$ *and its unique child is* $\mathtt{T}(\mathcal{G}')$.

- *If* $\mathcal{G} = \chi\langle\vec{e}\rangle$ *then* $\mathtt{T}(\mathcal{G})$ *consists of one node with label* $\chi\langle\vec{e}\rangle$.

- *If* $\mathcal{G} = \mathbf{0}$ *then* $\mathtt{T}(\mathcal{G})$ *consists of one node with label* $\mathbf{0}$.

We denote the set of assertion trees as $\mathcal{T}$ *and let* $T, T', \ldots$ *range over* $\mathcal{T}$.

Example 4.2. The assertion tree of $\mathcal{G}_{\text{ex4.1}}$ in Example 4.1 can be depicted as:



where identities of nodes are not shown and only their labels appear. ◇

For convenience, given $T \in \mathcal{T}$, we will use the partial functions

$$\mathtt{var}_T : \mathcal{N} \to 2^{\mathcal{V}}, \qquad \mathtt{cst}_T : \mathcal{N} \to \Psi, \qquad \text{and} \qquad snd_T, rcv_T : \mathcal{N} \to \mathbb{P}$$

that are undefined on $\mathcal{N}\backslash\{n \mid n \in T\}$ and defined as follows otherwise:

$$\mathtt{var}_T(n) = \begin{cases} \mathtt{var}(\iota), & \text{if } \underline{n} = \iota \\ \varnothing, & \text{otherwise} \end{cases} \qquad \mathtt{cst}_T(n) = \begin{cases} \psi, & \text{if } \underline{n} = \iota \text{ and } \mathtt{cst}(\iota) = \psi \\ \psi, & \text{if } \underline{n} = \{\psi\}\mathsf{I} \\ \textbf{true}, & \text{otherwise} \end{cases}$$

$$\mathtt{snd}_T(n) = \begin{cases} \mathtt{snd}(\iota), & \text{if } \underline{n} = \iota \\ \mathtt{s}, & \text{if } \underline{n} = \mathtt{s} \to \mathtt{r}: \end{cases} \qquad \mathtt{rcv}_T(n) = \begin{cases} \mathtt{rcv}(\iota), & \text{if } \underline{n} = \iota \\ \mathtt{r}, & \text{if } \underline{n} = \mathtt{s} \to \mathtt{r}: \end{cases}$$

Moreover, we shall use the following functions:

- $\mathtt{parent}_T(n)$ returning ε if $n = T^\bullet$, the parent of n in T if $n \in T$, and $\bot$ otherwise.
- $n{\uparrow_T}$ returning the path from $T^\bullet$ to n (including n) if $n \in T$, and $\bot$ otherwise.

Given $T \in \mathcal{T}$, let $\mathtt{A}(T)$ be the global assertion obtained by appending the labels of the nodes in (depth-first) preorder traversal visit of T.

Proposition 4.1. $\mathtt{A}(\mathtt{T}(\mathcal{G})) = \mathcal{G}$

Proposition 4.1 allows us to extend $\mathtt{knows}_\mathtt{p}(\_)$ to $\mathcal{T}$ by $\mathtt{knows}_\mathtt{p}(T) \stackrel{def}{=} \mathtt{knows}_\mathtt{p}(\mathtt{A}(T))$.

Proposition 4.2. *If* $T \in \mathcal{T}$ *then* $\mathtt{T}(\mathtt{A}(T)) = T$

Propositions 4.1 and 4.2 – whose proofs are trivial – basically induce an isomorphism between global assertions and their parsing trees.

4.3 On Recovering History Sensitivity

In a choreography, participants have to make local choices on the communicated values; such choices have an impact on the graceful coordination of the distributed parties. It is therefore crucial that the responsible party has "enough information" to commit to an

"appropriate" local choice, at each point of the choreography. For global assertions, this distills into *history sensitivity* (HS), a property defined in [12] as follows:

> A predicate guaranteed by a participant $\mathtt{s}$ can only contain those interaction variables that $\mathtt{s}$ knows.

HS demands the sender/selector of each interaction in a given assertion to know all the variables involved in the predicate associated to that interaction. We illustrate HS with the following example.

Example 4.3. The global assertion $\mathcal{G}_{\text{ex4.3}}$ below violates HS.

$$\begin{aligned}\mathcal{G}_{\text{ex4.3}} \quad = \quad & \mathtt{Alice} \to \mathtt{Bob}\colon \{v_1 \mid v_1 > 0\};\\ & \mathtt{Bob} \to \mathtt{Carol}\colon \{v_2 \mid v_2 > 0\};\\ & \mathtt{Carol} \to \mathtt{Alice}\colon \{v_3 \mid v_3 > v_1\}\end{aligned}$$

$\mathtt{Carol}$'s obligation $v_3 > v_1$ cannot be fulfilled because $v_1 \notin \mathtt{knows}_{\mathtt{Carol}}(\mathcal{G}_{\text{ex4.3}})$. ◇

Given a global assertion $\mathcal{G}$, the function $\overline{\mathtt{HS}}(\mathcal{G})$ below returns the nodes of $\mathtt{T}(\mathcal{G})$ where HS is violated

$$\overline{\mathtt{HS}}(\mathcal{G}) \stackrel{def}{=} \left\{ n \in \mathtt{T}(\mathcal{G}) \mid \mathtt{var}(\mathtt{cst}_T(n)) \nsubseteq \mathtt{knows}_{\mathtt{s}}(n\!\uparrow_T) \text{ and } \mathtt{s} = \mathtt{resp}_{\mathtt{T}(\mathcal{G})}(n) \right\}$$

where $\mathtt{resp}_T(\_) : \mathcal{N} \to \mathbb{P}$ yields the responsible party of a node and is defined as

$$\mathtt{resp}_T(n) \stackrel{def}{=} \begin{cases} \mathtt{snd}_T(n), & \text{if } \underline{n} = \iota \\ \mathtt{snd}_T(\mathtt{parent}_T(n)), & \text{if } \underline{n} = \{\psi\}\mathsf{I} \\ \bot, & \text{otherwise} \end{cases}$$

Intuitively, to determine whether a node $n \in \mathtt{T}(\mathcal{G})$ violates HS, one checks if the responsible party of n knows all the variables involved in $\mathtt{cst}_{\mathtt{T}(\mathcal{G})}(n)$.

Given $T \in \mathcal{T}$, $\mathtt{varHS}_T(\_)$ is a function from the nodes of T to sets of variables and is defined as follows.

$$\mathtt{varHS}_T(n) \stackrel{def}{=} \mathtt{var}(\mathtt{cst}_T(n)) \backslash \mathtt{knows}_{\mathtt{s}}(n\!\uparrow_T) \qquad \text{where} \quad \mathtt{s} = \mathtt{resp}_T(n)$$

Namely, $\mathtt{varHS}_T(n)$ yields the variables of n not known to the responsible party of n. It is a simple observation that if HS is violated at a node n, then there exists a variable in the predicate of n which is not known to the responsible party of n (in fact, $n \in \overline{\mathtt{HS}}(\mathcal{G})$ iff $\mathtt{varHS}_T(n) \neq \varnothing$).

Example 4.4. Consider the following global assertion:

$$\begin{aligned}
\mathcal{G}_{\text{ex4.4}} \quad = \quad & \mu\chi\langle 10\rangle\{v \mid v > 0\}. \\
& \quad \mathtt{Alice} \to \mathtt{Bob}\colon \{v_1 \mid v \geqslant v_1\}; \\
& \quad \mathtt{Bob} \to \mathtt{Carol}\colon \{v_2 \mid v_2 > v_1\}; \\
& \quad \mathtt{Carol} \to \mathtt{Alice}\colon \{v_3 \mid v_3 > v_1\}; \\
& \quad \mathtt{Carol} \to \mathtt{Bob}\colon \{v_4 \mid v_4 > v\}; \\
& \quad \chi\langle v_1\rangle
\end{aligned}$$

$\overline{\mathtt{HS}}(\mathcal{G}_{\text{ex4.4}}) = \{n_3, n_4\}$ where n_3 and n_4 are the nodes in $\mathtt{T}(\mathcal{G}_{\text{ex4.4}})$ corresponding to the third and fourth interactions of $\mathcal{G}_{\text{ex4.4}}$, i.e., $\underline{n_3} = \mathtt{Carol} \to \mathtt{Alice}\colon \{v_3 \mid v_3 > v_1\}$ and $\underline{n_4} = \mathtt{Carol} \to \mathtt{Bob}\colon \{v_4 \mid v_4 > v\}$.

Carol is responsible for both violations (i.e., $\mathtt{resp}_{\mathtt{T}(\mathcal{G}_{\text{ex4.4}})}(n_3) = \mathtt{resp}_{\mathtt{T}(\mathcal{G}_{\text{ex4.4}})}(n_4) =$ Carol). The violation in n_3 is on $\mathtt{varHS}_{\mathtt{T}(\mathcal{G}_{\text{ex4.4}})}(n_3) = \{v_1\}$ (i.e., Carol has to choose v_3 so that $v_3 > v_1$ without knowing v_1) and the violation in n_4 is on $\mathtt{varHS}_{\mathtt{T}(\mathcal{G}_{\text{ex4.4}})}(n_4) = \{v\}$ (i.e., Carol has to choose v_4 so that $v_4 > v$ without knowing v). Note that the violation of HS above does not imply that Carol will actually violate the condition $v_3 > v_1$. In fact, Carol could unknowingly choose either a violating or a non violating value for v_3. $\diamond$

In Section 4.3.1 and Section 4.3.2, we present two algorithms that fix, when possible, HS violations in a global assertion. We discuss and compare their applicability, as well as the relationship between the amended global assertion and the original one. We shall use Example 4.4 as the running example of Section 4.3.1 and Section 4.3.2.

4.3.1 Strengthening

Throughout this section we fix a global assertion $\mathcal{G}$ and its assertion tree $T = \mathtt{T}(\mathcal{G})$ and assume HS is violated at $n \in T$ with $\mathtt{cst}_T(n) = \psi$ and $\mathtt{resp}_T(n) = \mathtt{s}$.

Violations occur when the responsible party $\mathtt{s}$ of n ignores at least one variable $v \in \mathtt{var}(\psi)$. The strengthening algorithm (cf. Definition 4.4) replaces ψ in $\mathcal{G}$ with a predicate $\psi\{v'/v\}$ (if any) such that:

(*i*) v' is a variable that $\mathtt{s}$ knows,

(*ii*) if $\psi\{v'/v\}$ and the predicates occurring from $T^\bullet$ to $\mathtt{parent}_T(n)$ are satisfied, then ψ is also satisfied.

Intuitively, the method above *strengthens* ψ by replacing it with $\psi\{v'/v\}$ so that: due to (*i*) the presence of variable v, which is unknown to the sender/selector, is removed, and due to (*ii*) ψ can still be guaranteed. In fact, relying on the information provided by all the predicates occurring before n, if the sender/selector guarantees $\psi\{v'/v\}$ then they also guarantee ψ. If there is no variable v' that ensures (*i*) and (*ii*) then we say that *strengthening is not applicable*.

Let $\mathtt{PRED}_T : \mathcal{N} \to \Psi$ yield the conjunction of the predicates on the path from $T^\bullet$ to the

parent of a node:

$$\texttt{PRED}_T(n) \stackrel{def}{=} \begin{cases} \bot, & \text{if } \texttt{parent}_T(n) = \bot \\ \textbf{true}, & \text{if } \texttt{parent}_T(n) = \varepsilon \\ \texttt{cst}_T(\texttt{parent}_T(n)) \wedge \texttt{PRED}_T(\texttt{parent}_T(n)), & \text{otherwise} \end{cases}$$

The function $\texttt{strengthen}(\mathcal{G})$ uses $\texttt{PRED}_T$ to compute a global assertion $\mathcal{G}'$ by replacing in $\mathcal{G}$, if possible, the predicate violating HS by a stronger one.

Definition 4.3 (strengthen)**.** *The function* strengthen *is defined as follows.*

- *If* $\overline{\texttt{HS}}(\mathcal{G}) = \varnothing$ *then* $\texttt{strengthen}(\mathcal{G})$ *returns* $\mathcal{G}$.
- *If* $n \in \overline{\texttt{HS}}(\mathcal{G})$, $v \in \texttt{varHS}_T(n)$ *and there exists* $v' \in \texttt{knows}_\texttt{s}(n\!\uparrow_T)$ *such that*

$$\texttt{PRED}_T(n) \wedge \psi\{^{v'}/_{v}\} \implies \psi \quad \textit{with} \quad \psi = \texttt{cst}_T(n) \tag{4.3}$$

  then $\texttt{strengthen}(\mathcal{G})$ *returns* $\texttt{A}(T')$ *where* T' *is obtained from* T *by replacing* ψ *with* $\psi\{^{v'}/_{v}\}$ *in* $\underline{n}$.
- *When condition* (4.3) *does not hold for any* $v \in \texttt{knows}_\texttt{s}(n\!\uparrow_T)$, $\texttt{strengthen}(\mathcal{G})$ *returns* $\mathcal{G}^{\bot n}$, *indicating that* $\mathcal{G}$ *violates HS at* $n \in \overline{\texttt{HS}}(\mathcal{G})$.

Remark 4.4. *We assume that variables that are not under the scope of a quantifier are implicitly universally quantified.*

The algorithm Σ in Definition 4.4 below recursively applies $\texttt{strengthen}(\_)$ until either the global assertion satisfies HS or Σ is not applicable anymore.

Definition 4.4 (Σ)**.** *The algorithm* Σ *is defined as follows*

$$\Sigma(\mathcal{G}) \stackrel{def}{=} \begin{cases} \texttt{strengthen}(\mathcal{G}), & \textit{if } \texttt{strengthen}(\mathcal{G}) \in \{\mathcal{G}, \mathcal{G}^{\bot n}\} \\ \Sigma(\texttt{strengthen}(\mathcal{G})), & \textit{otherwise} \end{cases}$$

Example 4.5. Consider $\mathcal{G}_{\text{ex4.4}}$ from Example 4.4 and recall that $\overline{\texttt{HS}}(\mathcal{G}_{\text{ex4.4}}) = \{n_3, n_4\}$. Strengthening is applicable to n_3 since by substituting v_1 with v_2 in $v_3 > v_1$ (with $v_2 \in \texttt{knows}_{\texttt{Carol}}(n_3{\uparrow_{\texttt{T}(\mathcal{G}_{\text{ex4.4}})}})$) we have that condition (4.3) in Definition 4.3 holds:

$$(v > 0 \wedge v \geqslant v_1 \wedge v_2 > v_1) \wedge (v_3 > v_2) \implies (v_3 > v_1)$$

The invocation of $\texttt{strengthen}(\mathcal{G}_{\text{ex4.4}})$ returns (by substituting v_1 with v_2 in n_3):

$$\begin{aligned}
\mathcal{G}_{\text{ex4.5}} \quad = \quad & \mu\chi\langle 10\rangle\{v \mid v > 0\}. \\
& \quad \texttt{Alice} \to \texttt{Bob}\colon \{v_1 \mid v \geqslant v_1\}; \\
& \quad \texttt{Bob} \to \texttt{Carol}\colon \{v_2 \mid v_2 > v_1\}; \\
& \quad \texttt{Carol} \to \texttt{Alice}\colon \{v_3 \mid v_3 > v_2\}; \\
& \quad \texttt{Carol} \to \texttt{Bob}\colon \{v_4 \mid v_4 > v\}; \\
& \quad \chi\langle v_1\rangle
\end{aligned}$$

The invocation of $\texttt{strengthen}(\mathcal{G}_{\text{ex4.5}})$ returns $\mathcal{G}_{\text{ex4.5}}^{\perp n_4}$ since $\mathcal{G}_{\text{ex4.5}}$ has still one violating node n_4 for which strengthening is not applicable. In fact, $\texttt{knows}_{\texttt{Carol}}(n_4{\uparrow_{\texttt{T}(\mathcal{G}_{\text{ex4.5}})}}) = \{v_2, v_3\}$ and:

- by substituting v with v_2, condition (4.3) in Definition 4.3 does not hold since:

$$(v > 0 \wedge v \geqslant v_1 \wedge v_2 > v_1 \wedge v_3 > v_2) \wedge (v_4 > v_2) \implies (v_4 > v)$$

- by substituting v with v_3, condition (4.3) in Definition 4.3 does not hold since:

$$(v > 0 \wedge v \geqslant v_1 \wedge v_2 > v_1 \wedge v_3 > v_2) \wedge (v_4 > v_3) \implies (v_4 > v)$$

$\diamond$

4.3.2 Variable Propagation

An alternative approach to solve HS problems is based on the modification of global assertions by letting responsible parties of the violating nodes know the variables causing the violation. The idea is that such variables are propagated within a "chain of interactions".

Definition 4.5 ($\prec_T$-chain and propagation)**.** *Let $T \in \mathcal{T}$ and $n, n' \in T$, we write $n \prec_T n'$ iff n appears in $n'{\uparrow_T}$ and $\mathtt{rcv}_T(n) = \mathtt{snd}_T(n')$. A vector of nodes $n_1, \ldots, n_t$ is a $\prec_T$-*chain *iff $n_i \prec_T n_{i+1}$ for all $1 \leqslant i < t$.*

Given a $\prec_T$-chain $\vec{n} = n_1 \cdots n_t$ in T and $v_0 \in \mathtt{var}_T(n_t)$, let the propagation *of v_0 in $\vec{n}$ – written $\mathtt{P}_T(v_0, \vec{n})$ – be the tree T' obtained by updating the nodes in T as follows:*

- *for $1 \leqslant i < t$, $\mathtt{var}_{T'}(n_i) = \mathtt{var}_T(n_i)\, v_i$ and $\mathtt{cst}_{T'}(n_i) = \mathtt{cst}_T(n_i) \wedge (v_i = v_{i-1})$, with $v_1, \ldots, v_{t-1} \in \mathcal{V}$ fresh and pairwise distinct,*
- $\mathtt{cst}_{T'}(n_t) = \mathtt{cst}_T(n_t)\,\{^{v_{t-1}}/_{v_0}\}$,[3]
- *all the other nodes of T remain unchanged.*

Example 4.6. In the global assertion $\mathcal{G}_{\text{ex4.6}}$ below assume `Alice` knows v from previous interactions (the ellipsis in $\mathcal{G}_{\text{ex4.6}}$).

$$
\begin{array}{rcll}
\mathcal{G}_{\text{ex4.6}} & = & \ldots & \mathtt{Alice} \to \mathtt{Bob}\colon \{u_1 \mid \psi_1\}; \qquad (1)\\
 & & & \mathtt{Bob} \to \mathtt{Carol}\colon \{u_2 \mid \psi_2\}; \qquad (2)\\
 & & & \mathtt{Bob} \to \mathtt{Dave}\colon \{u_3 \mid \psi_3\}; \qquad (3)\\
 & & & \mathtt{Dave} \to \mathtt{Alice}\colon \{u_4 \mid u_4 > v\} \qquad (4)
\end{array}
$$

For the $\prec_T$-chain $\vec{n} = n_1\, n_3\, n_4$ in $\mathtt{T}(\mathcal{G}_{\text{ex4.6}})$ – where n_i corresponds to line (i) in $\mathcal{G}_{\text{ex4.6}}$ –

[3] Note that $t > 1$.

$\mathtt{P}_{\mathtt{T}(\mathcal{G}_{\text{ex4.6}})}(v,\vec{n})$ returns T' such that $\mathtt{A}(T')$ is

$$\begin{aligned}
\mathcal{G}'_{\text{ex4.6}} \quad = \quad \ldots \quad & \mathtt{Alice} \to \mathtt{Bob}\colon \{u_1\ v_1 \mid \psi_1 \wedge v = v_1\}; && (1)\\
& \mathtt{Bob} \to \mathtt{Carol}\colon \{u_2 \mid \psi_2\}; && (2)\\
& \mathtt{Bob} \to \mathtt{Dave}\colon \{u_3\ v_2 \mid \psi_3 \wedge v_1 = v_2\}; && (3)\\
& \mathtt{Dave} \to \mathtt{Alice}\colon \{u_4 \mid u_4 > v_2\} && (4)
\end{aligned}$$

$\diamond$

Hereafter, we fix a global assertion $\mathcal{G}$. Let $T = \mathtt{T}(\mathcal{G})$, $n \in \overline{\mathtt{HS}}(\mathcal{G})$, $v \in \mathtt{varHS}_T(n)$, and $\mathtt{s} = \mathtt{resp}_T(n)$. The *propagation* algorithm (cf. Definition 4.7) is *applicable* only if there exists a $\prec_T$-chain in $n{\uparrow_T}$ through which v can be propagated from a node whose sender knows v to a node where $\mathtt{s}$ is a receiver.

We define a function `propagate` which takes a global assertion $\mathcal{G}$ and returns $\mathcal{G}$ itself if HS is satisfied, $\mathcal{G}^{\perp n}$ if HS is violated at $n \in \mathtt{T}(\mathcal{G})$ and propagation is not applicable, and $\mathcal{G}'$ otherwise, where $\mathcal{G}'$ is obtained by propagating a violating variable v of node n. In the latter case, observe that v has necessarily been introduced in a node $n' \in n{\uparrow_{\mathtt{T}(\mathcal{G})}}$ from which v can be propagated, since we assume $\mathcal{G}$ closed.

Definition 4.6 (`propagate`)**.** *The function* $\mathtt{propagate}(\mathcal{G})$ *returns*

- $\mathcal{G}$*, if* $\overline{\mathtt{HS}}(\mathcal{G}) = \varnothing$*,*
- $\mathtt{P}_T(v,\vec{n})$*, if* $T = \mathtt{T}(\mathcal{G})$ *and there exists* $n \in \overline{\mathtt{HS}}(\mathcal{G})$*,* $v \in \mathtt{varHS}_T(n)$*, and* $\vec{n} = n_0\ \vec{n}_1\ n$ $\prec_T$*-chain in* T *such that* $\mathtt{snd}_T(n_0)$ *knows* v*,*
- $\mathcal{G}^{\perp n}$ *with* $n \in \overline{\mathtt{HS}}(\mathcal{G})$*, otherwise.*

Example 4.7. Consider again the global assertion $\mathcal{G}_{\text{ex4.5}}$ obtained after the invocation $\mathtt{strengthen}(\mathcal{G}_{\text{ex4.4}})$ in Example 4.5. In this case $\overline{\mathtt{HS}}(\mathcal{G}_{\text{ex4.5}}) = \{n_4\}$ with $\underline{n_4} = \mathtt{Carol} \to$

$\texttt{Bob}\colon \{v_4 \mid v_4 > v\}$. Propagation is applicable to n_4 and $\texttt{propagate}(\mathcal{G}_{\text{ex4.5}})$ returns

$$\begin{aligned}\mathcal{G}_{\text{ex4.7}} \quad = \quad & \mu\chi\langle 10\rangle\{v \mid v > 0\}.\\ & \texttt{Alice} \to \texttt{Bob}\colon \{v_1 \mid v \geqslant v_1\};\\ & \texttt{Bob} \to \texttt{Carol}\colon \{v_2\, u_1 \mid v_2 > v_1 \wedge u_1 = v\};\\ & \texttt{Carol} \to \texttt{Alice}\colon \{v_3 \mid v_3 > v_2\};\\ & \texttt{Carol} \to \texttt{Bob}\colon \{v_4 \mid v_4 > u_1\};\\ & \chi\langle v_1\rangle\end{aligned}$$

by propagating v from the second interaction – where the sender `Bob` knows v – to `Carol`, $\mathcal{G}_{\text{ex4.7}}$ satisfies HS. The predicate of the last interaction originates from the substitution $(v_4 > v)\{u_1/v\}$. ◇

The propagation algorithm is defined below and is based on a repeated application of $\texttt{propagate}(\_)$.

Definition 4.7 (Π)**.** *Given a global assertion $\mathcal{G}$, the function Π is defined as follows:*

$$\Pi(\mathcal{G}) = \begin{cases} \texttt{propagate}(\mathcal{G}), & \textit{if } \texttt{propagate}(\mathcal{G}) \in \{\mathcal{G}, \mathcal{G}^{\perp n}\} \\ \Pi(\texttt{propagate}(\mathcal{G})), & \textit{otherwise} \end{cases}$$

Remark 4.5. *In distributed applications it is often necessary to guarantee that exchanged information is accessible only to intended participants. It is worth observing that Π discloses information about the propagated variable to the participants involved in the propagation chain. The architect should therefore evaluate whether it is appropriate to use Π.*

One could think of an extension of $\texttt{propagate}(\mathcal{G})$ *which propagates variables only to participants entitled to know them. The existence of a propagation chain $\vec{n}$ for a variable v may be parametrised by two sets of participants chosen by the architect: a set A containing the participants who are allowed to know the value of v and a set N of participants*

not allowed to know it. Let $T = \mathtt{T}(\mathcal{G})$, and v_0 a variable causing an HS problem in $\mathcal{G}$, an acceptable chain $\vec{n}$ is defined as in Section 4.3.2 and such that

- *for all $n \in \vec{n}$, $\mathtt{rcv}_T(n) \notin N$, and*
- *there is no other $\prec_T$-chain $\vec{n}'$ for which $\mathtt{propagate}(\_)$ is applicable such that $|P(\vec{n}')| < |P(\vec{n})|$, where*

$$P(\vec{n}) = \{\mathtt{r} \mid \textit{there exist } n \in \vec{n} \textit{ such that } \mathtt{r} = \mathtt{rcv}_T(n) \textit{ and } \mathtt{r} \notin A\}$$

Note that even though this additional condition provides a more fine-grained control on the way problems are solved, it also decreases the range of applicability of the algorithm since the existence of such a chain is not guaranteed.

4.3.3 Properties of Σ and Π

We now discuss the properties of the global assertions amended by each algorithm and we compare them. Hereafter, we say Σ (resp. Π) returns $\mathcal{G}$ if it returns either $\mathcal{G}$ or $\mathcal{G}^{\perp n}$ for some n.

The applicability of Σ depends on whether it is possible to find a variable known to the responsible party of the violating node such that condition (4.3) in Definition 4.3 is satisfied. The applicability of Π depends on whether there exists a chain through which the problematic variable can be propagated. Observe also that there are cases in which Σ is applicable and Π is not, and vice versa, e.g., Example 4.5. Moreover, $\Sigma(\mathcal{G}) \neq \Pi(\mathcal{G})$ in general, hence it may not always be clear which one should be preferred.

We first show that our algorithms preserve the structure of the initial global assertion.

Lemma 4.1. *Let $\mathcal{G}$ be a global assertion;* $\mathtt{strengthen}(\mathcal{G})$ *(resp.* $\mathtt{propagate}(\mathcal{G})$*) always returns $\mathcal{G}'$ such that $\mathtt{T}(\mathcal{G}')$ is isomorphic to $\mathtt{T}(\mathcal{G})$ up-to labels, namely it has the same tree structure (but possibly different node labels).*

Proof. By Definition 4.3 $\texttt{strengthen}(\mathcal{G})$ always returns either $\mathcal{G}$ (which includes the case for $\mathcal{G}^{\perp n}$) or $\mathcal{G}'$. $\texttt{T}(\mathcal{G}')$ is isomorphic to $\texttt{T}(\mathcal{G})$ as the two trees either are the same or only differ in the label of one node.

By Definition 4.6, $\texttt{propagate}(\mathcal{G})$ always returns either $\mathcal{G}$ (which includes the case for $\mathcal{G}^{\perp n}$) or $\mathcal{G}' = \texttt{P}_{\texttt{T}(\mathcal{G})}(v_0, \vec{n})$ for some $\prec_T$-chain $\vec{n}$ and variable v_0. By definition of $\texttt{P}_{\texttt{T}(\mathcal{G})}(v_0, \vec{n})$, $\texttt{T}(\mathcal{G}')$ only differs from $\texttt{T}(\mathcal{G})$ in the labels of the nodes in $\vec{n}$, hence $\texttt{P}_{\texttt{T}(\mathcal{G})}(v_0, \vec{n})$ is isomorphic to $\texttt{T}(\mathcal{G})$. □

Proposition 4.3. *Let $\mathcal{G}$ be a global assertion. If it terminates, $\Sigma(\mathcal{G})$ (resp. $\Pi(\mathcal{G})$) returns $\mathcal{G}'$ such that $\texttt{T}(\mathcal{G}')$ is isomorphic to $\texttt{T}(\mathcal{G})$.*

Proof. By straightforward induction on the number of invocations of $\texttt{strengthen}(\mathcal{G})$ (resp. $\texttt{propagate}(\mathcal{G})$) in Σ (resp. Π), see [14] for more details. □

Lemma 4.2 below shows that the number of HS problems in a global assertion decreases after an invocation of either of our algorithm (if applicable).

Lemma 4.2. *Let $\mathcal{G}$ be a global assertion, $T = \texttt{T}(\mathcal{G})$, and k be the number of HS violations in T.*[4] *Let k_1, k_2 be the number of HS violations in $T_1 = \texttt{T}(\texttt{strengthen}(\mathcal{G}))$ and $T_2 = \texttt{T}(\texttt{propagate}(\mathcal{G}))$, respectively; then either $T_i = T$ or $k_i = k - 1$ with $i \in \{1, 2\}$.*

Proof. By Proposition 4.3, T and T_1 are isomorphic. Let $n_1 \in T_1$ be the node corresponding to $n \in T$. By definition of $\overline{\texttt{HS}}$ the violation in a node $n' \in T_1$ is defined only in terms of the nodes in $n'{\uparrow_{T_1}}$. By definition of $\texttt{strengthen}(\_)$ the only node from which T differs from T_1 is n_1. Hence, if a violation is added in T_1 with respect to T it must be in the subtree of T_1 rooted at n_1. However, a violation is not added in n_1 itself since the variable chosen to replace the problematic one is selected so that the responsible party *knows* it. No violation can be added in the subtree rooted at n' since $\texttt{strengthen}(\_)$ does not modify the variables known by the participants (but only the predicates). Thus, either $T_1 = T$ or $k_1 = k - 1$.

[4] Note that more than one violation may occur in one node if the sender does not know several variables.

In the case $T_2 = \mathtt{T}(\mathtt{propagate}(\mathcal{G}))$, assume $\hat{n} \in \overline{\mathtt{HS}}(T)$ with $\underline{\hat{n}} = \mathtt{s} \to \mathtt{r} : \{\vec{x} \mid \phi\}$, $v \in \mathtt{varHS}_T(\hat{n})$, and there exists an $\prec_T$-chain $n_0\, \vec{n}\, \hat{n}$ with $\underline{n_0} = \mathtt{s_0} \to \mathtt{r_0} : \{\vec{y_0} \mid \psi_0\}$ such that $\mathtt{s_0}$ knows v. We proceed by case analysis showing that, in any node $n \in T$, no violation is introduced, and exactly one violation is removed from $\hat{n}$.

- if $n = n_0$ no violation is introduced since $\underline{n_0}$ becomes $\mathtt{s_0} \to \mathtt{r_0} : \{\vec{y}\, v_0 \mid \psi_0 \wedge v_0 = v\}$ in T_2 where, by Definition 4.6, $\mathtt{s_0}$ knows v.

- if $n_i \in \vec{n}$ by definition of propagation $\underline{n_i} = \mathtt{s_i} \to \mathtt{r_i} : \{\vec{y_i} \mid \psi_i\}$ becomes $\mathtt{s_i} \to \mathtt{r_i} : \{\vec{y_i}\, v_i \mid \psi_i \wedge v_{i-1} = v_i\}$ in T_2 where, by Definition 4.5, $\mathtt{r_{i-1}} = \mathtt{s_i}$. It follows that $\mathtt{s_i}$ has previously received v_{i-1} hence $\mathtt{s_i}$ knows it.

- if $n = \hat{n}$, following Definition 4.5, $\mathtt{cst}_{T_2}(\hat{n}) = \phi\{v_0/v\}$ and the problem on v at $\hat{n}$ has been solved since $\mathtt{s}$ knows v_0, and v does not appear in ϕ anymore, hence $k_2 = k - 1$.

- if n does not belong to the $\prec_T$-chain $n_0\, \vec{n}\, \hat{n}$ then n remains unchanged and no violation is introduced. Note that if n is in the subtree rooted at $\hat{n}$, no violation is introduced since propagation does not decrease the knowledge of any participant. □

Lemma 4.2 implies directly that both algorithms terminate.

Corollary 4.1. *Let $\mathcal{G}$ be a global assertion; $\Sigma(\mathcal{G})$ and $\Pi(\mathcal{G})$ terminate.*

Whereas Σ does not change the global type underlying the global assertion, Π does. Indeed, in the resulting global assertion, more variables are exchanged in each interaction involved in the propagation. However, the structure of the tree remains the same.

Let $\mathtt{erase}(\mathcal{G})$ be the function that returns the underlying global type [44] corresponding to $\mathcal{G}$ (i.e., a global assertion without predicates).

Definition 4.8. *Given a global assertion $\mathcal{G}$, $\mathtt{erase}(\mathcal{G})$ is defined inductively as:*

$$\mathtt{erase}(\mathcal{G}) = \begin{cases} \mathtt{s} \to \mathtt{r} : \vec{\mathtt{a}}; \mathtt{erase}(\mathcal{G}') & \text{if } \mathcal{G} = \mathtt{s} \to \mathtt{r} : \{\vec{v} \mid \psi\}; \mathcal{G}' \\ \mathtt{s} \to \mathtt{r} : \left\{ \mathtt{l_j} : \mathtt{erase}(\mathcal{G}_j) \right\}_{j \in J} & \text{if } \mathcal{G} = \mathtt{s} \to \mathtt{r} : \left\{ \{\psi_j\} \mathtt{l_j} : \mathcal{G}_j \right\}_{j \in J} \\ \mu\chi.\mathtt{erase}(\mathcal{G}') & \text{if } \mathcal{G} = \mu\chi\langle\vec{e}\rangle\{\vec{v} \mid \psi\}.\, \mathcal{G}' \\ \mathcal{G} & \text{if } \mathcal{G} = \mathbf{0} \text{ or } \mathcal{G} = \chi \end{cases}$$

where $\vec{\mathtt{a}}$ is a list of sorts, so that the length of $\vec{v}$ is the same as $\vec{\mathtt{a}}$. If $\vec{\mathtt{a}} = \mathtt{a}_1 \ldots \mathtt{a}_n$ and $\vec{v} = v_1 \ldots v_n$, then for $1 \leqslant i \leqslant n$, $\mathtt{a}_i$ is the sort of variable v_i.

Lemma 4.3. *Let $\mathcal{G}$ and $\mathcal{G}'$ be two assertions that differ only in the predicates annotating interactions and branching constructs. Then $\mathtt{erase}(\mathcal{G}) = \mathtt{erase}(\mathcal{G}')$.*

Proof. The proof is by straightforward structural induction on $\mathcal{G}$ (cf. [14]). □

Lemma 4.4. *Let $\mathcal{G}$ be a global assertion and $T = \mathtt{T}(\mathcal{G})$. Given a $\prec_T$-chain $\vec{n} = n_1, \ldots, n_t$ in T, if $\mathtt{P}_T(v_0, \vec{n}) = \mathcal{G}'$ for an interaction variable v_0, then for all $n \in \mathtt{T}(\mathcal{G})$ and its corresponding node $n' \in \mathtt{T}(\mathcal{G}')$,*

$$\mathtt{var}_{\mathtt{T}(\mathcal{G})}(n) \subseteq \mathtt{var}_{\mathtt{T}(\mathcal{G}')}(n')$$

Proof. Observe that, by Proposition 4.3, T is isomorphic to T'. For each $n \in T$, let n' denote the node in T' that corresponds to n; and let $\vec{n} = n_1, \ldots, n_t$. We proceed by case analysis on the (labels of the) nodes of T, which we divide in three groups according to the definition of $\mathtt{P}_T(v_0, \vec{n})$:

- for $i \in \{1, \ldots, t-1\}$, $\mathtt{var}_T(n_i) = \mathtt{var}_{T'}(n'_i)\ v_i$.
- $\mathtt{var}_T(n_t) = \mathtt{var}_{T'}(n'_t)$.
- if $n \in T$ and $n \notin \vec{n}$, then n' has the same label as n since by definition of $\mathtt{P}_T(v_0, \vec{n})$ all the nodes that are not in $\vec{n}$ are unchanged. □

Proposition 4.4 (Underlying type structure)**.** *Let $\mathcal{G}$ be a global assertion,*

- *if* $\Sigma(\mathcal{G})$ *returns* $\mathcal{G}'$ *then* $\texttt{erase}(\mathcal{G}) = \texttt{erase}(\mathcal{G}')$
- *if* $\Pi(\mathcal{G})$ *returns* $\mathcal{G}'$ *then for all* $n \in \texttt{T}(\mathcal{G})$ *and its corresponding node* $n' \in \texttt{T}(\mathcal{G}')$*,*

$$\texttt{var}_{\texttt{T}(\mathcal{G})}(n) \subseteq \texttt{var}_{\texttt{T}(\mathcal{G}')}(n')$$

Proof. Consider Σ; we observe that $\texttt{strengthen}(\mathcal{G})$ either returns $\mathcal{G}$ or a $\mathcal{G}'$ that differs from $\mathcal{G}$ only in the predicate of one (interaction or branching) node. By Lemma 4.3:

$$\texttt{erase}(\mathcal{G}) = \texttt{erase}(\texttt{strengthen}(\mathcal{G}))$$

The rest follows by straightforward induction on the number of applications of Σ, see [14] for more details.

Consider Π; we observe that, by Lemma 4.4, if $\texttt{propagate}(\mathcal{G}) = \mathcal{G}'$ then for all $n \in \texttt{T}(\mathcal{G})$ and its corresponding node $n' \in \texttt{T}(\mathcal{G}')$,

$$\texttt{var}_{\texttt{T}(\mathcal{G})}(n) \subseteq \texttt{var}_{\texttt{T}(\mathcal{G}')}(n') \tag{4.4}$$

In fact, $\texttt{propagate}(\mathcal{G})$ either returns $\mathcal{G}' = \mathcal{G}$ or $\mathcal{G}' = \texttt{P}_{\texttt{T}(\mathcal{G})}(v_0, \vec{n})$. In the former case (4.4) holds trivially, in the latter case it holds by Lemma 4.4. The rest follows by straightforward induction on the number of applications of Π (cf. [14]). □

The application of Σ and Π affects the predicates of the original global assertion. In Σ, strengthening allows less values for the interaction variables of the amended interaction. Conversely, the predicates computed by Π are equivalent to the original ones (i.e., they allow sender and receiver to choose/expect the same set of values). Nevertheless, such predicates are syntactically different as Π adds the equality predicates on the propagated variables.

Proposition 4.5 (Assertion predicates)**.** *Let $\mathcal{G}$ be a global assertion,*

1. *if $\Sigma(\mathcal{G})$ returns $\mathcal{G}'$ then for all $n \in \mathtt{T}(\mathcal{G})$ whose label is modified by Σ, and its corresponding node $n' \in \mathtt{T}(\mathcal{G}')$ (cf. Proposition 4.4), it holds that*

$$\mathtt{PRED}_{\mathtt{T}(\mathcal{G}')}(n') \wedge \mathtt{cst}_{\mathtt{T}(\mathcal{G}')}(n') \implies \mathtt{cst}_{\mathtt{T}(\mathcal{G})}(n)$$

2. *if $\Pi(\mathcal{G})$ returns $\mathcal{G}'$ then for all $n \in \mathtt{T}(\mathcal{G})$ whose label is modified by Π, and its corresponding node $n' \in \mathtt{T}(\mathcal{G}')$*

   (a) $\mathtt{cst}_{\mathtt{T}(\mathcal{G}')}(n')$ *is the predicate* $\mathtt{cst}_{\mathtt{T}(\mathcal{G})}(n)\sigma \wedge \psi$

   (b) $\mathtt{PRED}_{\mathtt{T}(\mathcal{G})}(n) \wedge \mathtt{cst}_{\mathtt{T}(\mathcal{G})}(n) \wedge \psi \iff \mathtt{PRED}_{\mathtt{T}(\mathcal{G}')}(n') \wedge \mathtt{cst}_{\mathtt{T}(\mathcal{G}')}(n')$

   For some satisfiable $\psi \in \Psi$ and variable substitution σ.

Proof. **Item 1.** The proof relies on the fact that Σ either does not change $\mathcal{G}$ or replaces a problematic variable by a variable for which (4.3) holds. We show the result by showing that it holds for each invocation of $\mathtt{strengthen}(\_)$ by Σ. Indeed, for each invocation we have that, by Definition 4.3, if n is modified by Σ, then we have that $n \in \overline{\mathtt{HS}}(\mathtt{T}(\mathcal{G}))$. In addition, there must be $v \in \mathtt{varHS}_{\mathtt{T}(\mathcal{G})}(n)$ such that there exists $v' \in \mathtt{knows}_{\mathtt{s}}(n\uparrow_{\mathtt{T}(\mathcal{G})})$ and (4.3) is satisfied. This gives us

$$\underbrace{\mathtt{PRED}_{\mathtt{T}(\mathcal{G})}(n)}_{\mathtt{PRED}_{\mathtt{T}(\mathcal{G}')}(n')} \wedge \underbrace{\psi\{v'/v\}}_{\mathtt{cst}_{\mathtt{T}(\mathcal{G}')}(n')} \implies \underbrace{\psi}_{\mathtt{cst}_{\mathtt{T}(\mathcal{G})}(n)}$$

Since only the predicate of node n is updated by substituting v by v', by Definition 4.3.

Item 2. The proof relies on the definition of $\mathtt{P}_{\mathtt{T}(\mathcal{G})}(\_,\_)$, i.e., a predicate of the form $v_1 = v_0$ or $v_i = v_{i-1}$ is added to each predicate of the nodes in the chain, and problematic variables are replaced by fresh ones. The additional predicates are satisfiable since

they constrain only fresh variables (i.e., v_i). We have these results by showing that each invocation of $\texttt{propagate}(\_)$ by Π validates the result.

Item 2a. If n is modified by $\texttt{propagate}(\_)$, then $n \in \vec{n}$, by Definition 4.6. Assume v_0 is the variable to be propagated.

- If n is not the last node of $\vec{n}$, by definition of $\texttt{P}_{\texttt{T}(\mathcal{G})}(v_0, \vec{n})$, we have that $\texttt{cst}_{\texttt{T}(\mathcal{G}')}(n') = \texttt{cst}_{\texttt{T}(\mathcal{G})}(n) \wedge (v_i = v_{i-1})$, which gives us the expected result if ψ is $v_i = v_{i-1}$ and σ is the empty substitution.

- If n is the last node of $\vec{n}$, by definition of $\texttt{P}_{\texttt{T}(\mathcal{G})}(v_0, \vec{n})$, we have that $\texttt{cst}_{\texttt{T}(\mathcal{G})}(n') = \texttt{cst}_{\texttt{T}(\mathcal{G})}(n)\{^{v_{t-1}}/_{v_0}\}$, which gives the expected result with $\sigma = \{^{v_{t-1}}/_{v_0}\}$ and $\psi = \textbf{true}$.

Item 2b. If n is modified by $\texttt{propagate}(\_)$, then $n \in \vec{n}$, by Definition 4.6. Assume v_0 is the variable to be propagated and the length of $\vec{n}$ is k.

- If n is the first node of $\vec{n}$, then

$$\texttt{cst}_{\texttt{T}(\mathcal{G}')}(n') = \texttt{cst}_{\texttt{T}(\mathcal{G})}(n) \wedge (v_1 = v_0) \quad \text{and} \quad \texttt{PRED}_{\texttt{T}(\mathcal{G}')}(n') = \texttt{PRED}_{\texttt{T}(\mathcal{G})}(n)$$

  since $n'{\uparrow}_{\texttt{T}(\mathcal{G}')}$ is unchanged. We have the expected result if ψ is $v_1 = v_0$. Note that by definition of $\texttt{P}_{\texttt{T}(\mathcal{G})}(v_0, \vec{n})$, v_1 is a fresh variable therefore $v_1 = v_0$ is satisfiable.

- If n is the i^{th} node in $\vec{n}$ $(1 < i < k)$ then $\texttt{cst}_{\texttt{T}(\mathcal{G}')}(n') = \texttt{cst}_{\texttt{T}(\mathcal{G})}(n) \wedge (v_i = v_{i-1})$, and

$$\texttt{PRED}_{\texttt{T}(\mathcal{G}')}(n') = \texttt{PRED}_{\texttt{T}(\mathcal{G})}(n) \wedge \bigwedge_{1<j<i} v_j = v_{j-1}$$

  where each $v_j = v_{j-1}$ is satisfiable since each variable is freshly introduced. We have the expected result with ψ as $v_i = v_{i-1}$.

- If n is the last node in $\vec{n}$, then $\mathtt{cst}_{\mathtt{T}(\mathcal{G}')}(n') = \mathtt{cst}_{\mathtt{T}(\mathcal{G})}(n)\{^{v_k}/_{v_0}\}$, and

$$\mathtt{PRED}_{\mathtt{T}(\mathcal{G}')}(n') = \mathtt{PRED}_{\mathtt{T}(\mathcal{G})}(n) \wedge \bigwedge_{1<j<k} v_j = v_{j-1}$$

  with each $v_j = v_{j-1}$ satisfiable, as before. We have the required result with ψ set to **true**. □

The statement 2 (b) in Proposition 4.5 amounts to saying that $\mathtt{cst}_{\mathtt{T}(\mathcal{G})}(n) \wedge \psi$ is equivalent to the predicate $\mathtt{cst}_{\mathtt{T}(\mathcal{G}')}(n')$ when considered in its respective context.

Remarkably, Σ and Π do not add violations (of either HS or TS) to the amended global assertions. We postpone the discussion of this property to Section 4.4.3 (Proposition 4.7) after the formal introduction of TS.

Finally, we prove that if the value returned by Σ or Π is not of the type $\mathcal{G}^{\perp n}$ then the amended global assertion satisfies HS.

Theorem 4.1 (Correctness)**.** *If there is* $\mathcal{G}'$ *such that* $\Sigma(\mathcal{G}) = \mathcal{G}'$ *or* $\Pi(\mathcal{G}) = \mathcal{G}'$ *then* $\overline{\mathtt{HS}}(\mathcal{G}') = \varnothing$.

Proof. **Case Σ.** By Definitions 4.3 and 4.4, Σ terminates successfully when $\overline{\mathtt{HS}}(\mathcal{G}) = \varnothing$. We show that at each iteration of Σ, the number of HS violations decreases. Assume that there is k violations in $\mathcal{G}$, by Definition 4.4, we have either

- $\Sigma(\mathcal{G}) = \mathtt{strengthen}(\mathcal{G}) = \mathcal{G}$ in which case, by Definition 4.3, $\overline{\mathtt{HS}}(\mathcal{G}) = \varnothing$, i.e., $k = 0$, and the function terminates, or

- $\Sigma(\mathcal{G}) = \Sigma(\mathtt{strengthen}(\mathcal{G})) = \mathcal{G}'$ with $\mathcal{G} \neq \mathcal{G}'$, and by Lemma 4.2 the number of HS violation in $\mathtt{strengthen}(\mathcal{G})$ is strictly less than k.

Case Π. The case for Π is similar to the previous case, using Definition 4.6 (resp. 4.7) instead of Definition 4.3 (resp. 4.4). □

4.4 On Recovering Temporal Satisfiability

In a choreography, the local choices made by some parties may restrict later choices of other parties to the point that no suitable value is available. This would lead to an abnormal termination since the choreography cannot continue. For global assertions, this distills into *temporal satisfiability* (TS) which requires that the values sent in each interaction do not compromise the satisfiability of future interactions. The formal definition of temporal satisfiability is adapted from [12].

Definition 4.9 (TS [12])**.** *A global assertion* $\mathcal{G}$ satisfies TS*, written* $\mathtt{TS}(\mathcal{G})$*, if and only if* $GSat(\mathcal{G}, \mathbf{true})$ *holds where*

$$GSat(\mathcal{G},\psi)\ \textit{iff}\begin{cases} GSat(\mathcal{G}',\psi \wedge \mathtt{cst}(\iota)), & \textit{if } \mathcal{G} = \iota;\mathcal{G}' \textit{ and } \psi \implies \exists\, \mathtt{var}(\iota) : \mathtt{cst}(\iota) \\ \bigwedge_{j\in J} GSat(\mathcal{G}_j,\psi \wedge \psi_j), & \textit{if } \mathcal{G} = \mathtt{s} \to \mathtt{r} : \{\{\psi_j\}\mathsf{l}_j : \mathcal{G}_j\}_{j\in J} \textit{ and } \psi \implies \bigvee_{j\in J} (\psi_j) \\ GSat(\mathcal{G}',\psi \wedge \psi'), & \textit{if } \mathcal{G} = \mu\chi\langle\vec{e}\rangle\{\vec{v} \mid \psi'\}.\,\mathcal{G}' \textit{ and } \psi \implies \psi'\{\vec{e}/\vec{v}\} \\ GSat(\mathcal{G}',\psi \wedge \psi'), & \textit{if } \mathcal{G} = \chi_{\psi'(\vec{v})}\langle\vec{e}\rangle \textit{ and } \psi \implies \psi'\{\vec{e}/\vec{v}\} \\ \mathcal{G} = \mathbf{0}, & \textit{otherwise} \end{cases}$$

For an assertion tree $T \in \mathcal{T}$*,* $\mathtt{TS}(T)$ *holds iff* $GSat(\mathtt{A}(T), \mathbf{true})$.

Intuitively, the predicate ψ in Definition 4.9 is the conjunction of all the predicates that precede an interaction in $\mathcal{G}$. In the first case, all the values satisfying ψ allow to instantiate the interaction variables $\mathtt{var}(\iota)$ so to satisfy the constraint $\mathtt{cst}(\iota)$ of ι. For branching, *GSat* requires that at least one branch can be chosen and that each possible path satisfies *GSat*. For recursive definition, we require that the initial parameters satisfy the invariant ψ'. We assume that each recursive call is annotated with the invariant of its corresponding

recursive definition, i.e., in Definition 4.9, $\psi'(\vec{v})$ is the predicate corresponding to the invariant of the definition of χ.

Often, TS problems appear when one tries to restrict the domain of a variable after its introduction. To illustrate this, we introduce the following running example.

Example 4.8. Consider $\mathcal{G}_{\text{ex4.8}}$ below, where $\mathtt{s}$ constrains x and y:

$$\begin{aligned}\mathcal{G}_{\text{ex4.8}} \quad = \quad & \mathtt{s} \to \mathtt{r}\colon \{x \mid x < 10\};\\ & \mathtt{s} \to \mathtt{r}\colon \{y \mid y > 8\};\\ & \mathtt{r} \to \mathtt{s}\colon \{z \mid x > z \wedge\ z > 6\ \wedge\ y \neq z\}\end{aligned}$$

When $\mathtt{r}$ introduces z, both x and y are further restricted. $\mathcal{G}_{\text{ex4.8}}$ violates TS because it does not hold that (cf. Definition 4.9)

$$\forall xy\colon \quad \underbrace{(x<10)\wedge(y>8)}_{\psi} \quad \Longrightarrow \quad \underbrace{\exists z\colon}_{\exists\,\mathtt{var}(\iota)\colon} \quad \underbrace{(x>z\wedge\ z>6\ \wedge\ y\neq z)}_{\mathtt{cst}(\iota)}$$

Noticeably, if $\mathtt{s}$ chooses, e.g., $x = 6$ then $\mathtt{r}$ cannot choose a value for z. ◇

Possibly, TS can be regained by rearranging some predicates. In particular, we can "lift" a predicate to a previous interaction node. For instance, in Example 4.8, one could lift the predicate $\exists z\colon\ x > z > 6$ (adapted from the last interaction) to the first interaction's predicate.

We first consider TS violations occurring in interactions and recursive definitions. Amending violations arising in branching and recursive calls is similar but complicates the presentation; for the sake of clarity, such violations are considered in Section 4.4.2.

4.4.1 Lifting Algorithm

We formalise the lifting algorithm. First, we give a function telling us whether a *node n* violates TS.

Definition 4.10 (TSnode)**.** *Given $T \in \mathcal{T}$, $\texttt{TSnode}_T(n)$ holds iff $n \in T$ and $\texttt{TS}(n{\uparrow_T})$ holds. In addition, we assume that* TSnode *holds for nodes with label of the form* $\texttt{s} \to \texttt{r}:$ *(since there is no predicate in these nodes, no TS problem can arise).*

We now define a function that returns a set of nodes violating TS such that all the previous nodes in the tree do not violate TS.

Definition 4.11 ($\overline{\texttt{TS}}$)**.** *The function $\overline{\texttt{TS}} : \mathcal{T} \to \mathcal{N}$ is defined as follows:*

$$\overline{\texttt{TS}}(T) \stackrel{def}{=} \left\{ n \in T \;\middle|\; \begin{array}{c} \texttt{TSnode}_T(n) \textit{ is false and } \texttt{TSnode}_T(n') \\ \textit{is true for all } \; n' \in \texttt{parent}_T(n){\uparrow_T} \end{array} \right\}$$

For instance, in Example 4.8, we have that $\overline{\texttt{TS}}(T_{\text{ex4.8}})$ is the singleton $\{n_{\text{ex4.8}}\}$ where $T_{\text{ex4.8}} = \texttt{T}(\mathcal{G}_{\text{ex4.8}})$ and $n_{\text{ex4.8}}$ is the node corresponding to the last interaction of $\mathcal{G}_{\text{ex4.8}}$.

Once an *interaction* node $n \in \overline{\texttt{TS}}(T)$ is chosen, the next step is to identify which part of its predicate is the source of the problem. To this purpose, we define a relation among predicates ψ and ϕ in a context ψ'.

Definition 4.12 (Conflict)**.** *The predicate $\psi \in \Psi$* is in conflict *on $\vec{v} \subseteq \mathcal{V}$ with ϕ* in *ψ' iff*

$$\psi' \implies \exists \vec{v} : \phi \quad \textit{and} \quad \psi' \not\Longrightarrow \exists \vec{v} : (\phi \wedge \psi)$$

The notion of conflict is based on the definition of TS for interaction nodes (Definition 4.9). On the one hand, the part ϕ of the predicate does not spoil TS and, on the other hand, the part ψ, in conjunction with ϕ, invalidates TS.

Example 4.9. In Example 4.8, we have

$$\forall xy : \; x < 10 \wedge y > 8 \implies \exists z : \; y \neq z$$

and

$$\forall xy : \; x < 10 \wedge y > 8 \not\Longrightarrow \exists z : \; y \neq z \; \wedge \; x > z \wedge \; z > 6$$

That is to say that $x > z \wedge\ z > 6$ is the "problematic" part of the predicate. $\diamond$

Using Definition 4.12 and $\texttt{PRED}_T(n)$ (cf. Section 4.3.1), we define

$$\texttt{split}_T(n,\psi) \stackrel{def}{=} \left\{ \psi' \;\middle|\; \psi \iff \psi' \wedge \phi \text{ and } \psi' \textit{ is in conflict} \text{ on } \texttt{var}(n) \text{ with } \phi \textit{ in } \texttt{PRED}_T(n) \right\}$$

which returns a set of problematic predicates. Note that $\iff$ denotes a logical equivalence between ψ and $\psi' \wedge \phi$; we assume that ψ' and ϕ are sub-formulae of ψ.

Example 4.10. Considering Examples 4.8 and 4.9, the application of `split` yields

$$\texttt{split}_{T_{\text{ex4.8}}}(n_{\text{ex4.8}}, z > 6 \wedge x > z \wedge y \neq z) = \{z > 6 \wedge x > z\}$$

since $y \neq z$ allows to choose a suitable value for z. $\diamond$

Remark 4.6. *For a tree $T \in \mathcal{T}$ and $n \in \overline{\texttt{TS}}(T)$ such that $\psi' \in \texttt{split}_T(n,\psi)$, we may have $\texttt{PRED}_T(n) \nRightarrow \exists \vec{v} : \psi'$. For instance, if the predicate ψ' is not satisfiable, e.g., $\psi' = v < 7 \wedge v > 7$. In this case the lifting algorithm is not applicable.*

Remark 4.7. *Note that at this level, it is not necessary to require ψ' to be minimal in the definition of* `split` *(in terms of, e.g., the size of the formula or the number of variables in ψ'). Indeed, as stated later in Definition 4.14, only the predicates which can be lifted successfully are used by the algorithm. However, an implementation of the algorithm could minimise the predicate in order to maximise the efficiency of the lifting algorithm.*

The next definition formalises the construction of a new assertion tree which possibly regains TS, given a node and an assertion to be "lifted" (i.e., a "problematic" predicate).

Definition 4.13 (`build`)**.** *The function $\texttt{build}_T(n,\psi)$ returns:*

- *$\hat{T} \in \mathcal{T}$, if ψ is satisfiable and we can construct $\hat{T}$ isomorphic to T except that, each*

node

$$n' \in (\mathtt{parent}_T(n))\!\uparrow_T \ \textit{such that}\ \underline{n'} = \mathtt{s} \to \mathtt{r}\colon \{\vec{u} \mid \theta\}\ \textit{with}\ \vec{u} \cap \mathtt{var}(\psi) \neq \varnothing$$

is replaced by a node $\hat{n}$ with label

$$\mathtt{s} \to \mathtt{r}\colon \{\vec{u} \mid \theta \ \wedge\ \forall \vec{x}\colon\ \exists \vec{y}\colon\ \psi\} \quad \textit{such that} \quad \theta \wedge \forall \vec{x}\colon\ \exists \vec{y}\colon\ \psi\ \textit{is satisfiable}$$

where

- $\vec{x} \subseteq \mathtt{var}(\psi) \backslash \mathtt{knows}_{\mathtt{s}}(T)$ *are introduced in a node in* $n'\!\uparrow_T$
- $\vec{y} \subseteq \mathtt{var}(\psi)$ *are introduced in a node in the subtree rooted at* n'

and there is no $n'' \in (\mathtt{parent}_T(n))\!\uparrow_T$ *such that* $\underline{n''} = \mu\chi\langle\vec{e}\rangle\{\vec{v} \mid \psi'\}$ *with* $\vec{v} \cap \mathtt{var}(\psi) \neq \varnothing$.

- $\perp$ *otherwise.*

Essentially, Definition 4.13 duplicates a quantified version of the predicate ψ in the nodes which introduce variables in $\mathtt{var}(\psi)$. For each updated node n', the quantification of the variables in $\mathtt{var}(\psi)$ operates in the following way. The variables which are introduced before n' in the tree and which are not known to s are quantified *universally* (since s has no control over them). The variables that are introduced later in the tree are quantified *existentially*, so that s may choose values for the variables in $\vec{u}$ which do not compromise the satisfiability of predicates down in the tree.

Remark 4.8. *In the definition of* `build`*, we assume that if either $\vec{x}$ or $\vec{y}$ is empty, the corresponding unnecessary quantifier is removed. Recall that global assertions are closed (cf. Section 4.2). Therefore all the variables in* $\mathtt{var}(\psi)$ *are either quantified in the predicate of $\hat{n}$, or have been introduced before n'.*

Continuing from Example 4.8, we would invoke $\texttt{build}_{T_{\text{ex4.8}}}(n_{\text{ex4.8}}, z > 6 \wedge x > z)$ which returns a new assertion tree. The new tree can be transformed into a global assertion isomorphic to $\mathcal{G}_{\text{ex4.8}}$ with the first line updated to: $\texttt{s} \to \texttt{r}\colon \{x \mid x < 10 \wedge \exists z : \ x > z > 6\}$.

The function $\texttt{TSres} : \mathcal{T} \times \mathcal{N} \to \mathcal{T} \cup \bot$ brings the above definitions together in order to either fix a TS problem at n, or return $\bot$.

Definition 4.14 (TSres)**.** *Given $T \in \mathcal{T}$ and $n \in \overline{\texttt{TS}}(T)$, we define*

$$\texttt{TSres}_T(n) = \begin{cases} \texttt{build}_T(n, \psi), & \text{if } \underline{n} = \iota \text{ and there is } \psi \in \texttt{split}_T(n, \texttt{cst}_T(n)) \\ & \text{s.t. } \texttt{build}_T(n, \psi) \neq \bot \\ \texttt{build}_T(n, \psi\{\vec{e}/\vec{v}\}), & \text{if } \underline{n} = \mu\chi\langle\vec{e}\rangle\{\vec{v} \mid \psi\} \\ \bot, & \text{otherwise} \end{cases}$$

The first case of Definition 4.14 handles TS problems in interaction nodes. If there is a predicate ψ in conflict such that it can be "lifted" by build successfully, then the function returns the result of build. The second case handles TS violations in recursive definitions. The problem is similar to the interaction case, but in this case, the values assigned to the recursion parameters are known (i.e., $\vec{e}$). It may be possible to lift the recursion invariant, where we replace the recursion parameters by the corresponding initialisation vector. Example 4.11 illustrates this case.

Example 4.11. For the global assertion $\mathcal{G}_{\text{ex4.11}}$ given below, $\texttt{TS}(\mathcal{G}_{\text{ex4.11}})$ does not hold because $\forall x\, y : \ \textbf{true} \implies (x > y > 6)$.

$$\begin{aligned} \mathcal{G}_{\text{ex4.11}} \quad = \quad & \texttt{s} \to \texttt{r}\colon \{x \mid \textbf{true}\}; \\ & \mu\chi\langle 8\rangle\{y \mid x > y > 6\}.\, \mathcal{G}' \end{aligned}$$

However, using the initialisation parameters, we can lift $x > 8 > 6$, i.e., the original predicate where we replaced y by 8, to the interaction preceding the recursion. TS now holds

in the new global assertion (assuming that $\texttt{TS}(\mathcal{G}')$ holds as well). $\diamond$

Remark 4.9. *In Example 4.11, if we had only lifted $x > y > 6$, as in the interaction case, it would not have solved the TS problem. Indeed, the predicate of the first interaction would have become $\exists y : x > y > 6$ which does not exclude values for x which are incompatible with the invariant (e.g., $x = 8$).*

The overall lifting procedure relies on a repeated application of `TSres` until either the assertion tree validates TS or the function fails to solve the problem. In the latter case, the function returns the most improved version of the tree and the node at which it failed.

Definition 4.15 (Λ)**.** Λ *is defined as follows, given a global assertion $\mathcal{G}$.*

$$\Lambda(\mathcal{G}) = \begin{cases} \mathcal{G}, & \textit{if } \texttt{TS}(\mathcal{G}) \\ \Lambda(\texttt{TSres}_{\texttt{T}(\mathcal{G})}(n)), & \textit{if there is } n \in \overline{\texttt{TS}}(\texttt{T}(\mathcal{G})) \textit{ s.t. } \texttt{TSres}_{\texttt{T}(\mathcal{G})}(n) \neq \bot \\ \mathcal{G}^{\bot n}, & \textit{otherwise} \end{cases}$$

4.4.2 Applying Λ to Branching and Recursion

Branching. The underlying idea of branching is to enable the architect to design a choreography where a branch cannot be taken when some variables have a particular value. The architect should be involved in the resolution of the problem, because two options are possible; either the disjunction of all the predicates found in the branches is lifted, or one of the branches predicate is lifted. Arguably, the latter may also prohibit other branches to be chosen, as shown in Example 4.12 below.

Observe that according to Definition 4.9, TS fails on branching nodes only when there are values for which none of the predicates decorating the branches are satisfiable.

Example 4.12. As an illustration, we consider the following global assertion:

$$\begin{aligned} \mathcal{G}_{\text{ex4.12}} \quad = \quad & \mathtt{r} \to \mathtt{s} \colon \{v \mid \mathbf{true}\}; \\ & \mathtt{s} \to \mathtt{r} \colon \left\{ \{v > 5\}\, l_1 : \mathcal{G}_1, \right. \\ & \qquad \left. \{v < 5\}\, l_2 : \mathcal{G}_2 \right\} \end{aligned}$$

Assuming that $\mathtt{TS}(\mathcal{G}_1)$ and $\mathtt{TS}(\mathcal{G}_2)$ hold, we have that $\mathtt{TS}(\mathcal{G}_{\text{ex4.12}})$ does not hold because $\mathbf{true} \implies (v > 5 \vee v < 5)$. Clearly, if $v = 5$ then no branch may be selected.

Let us call $\hat{n}$ the node corresponding to the branching in the second line of $\mathcal{G}_{\text{ex4.12}}$. Depending on the intention of the architect the problem could be fixed by one of these invocations to `build` (where, in both cases, superfluous quantifiers are removed).

- $\mathtt{build}_{\mathtt{T}(\mathcal{G}_{\text{ex4.12}})}(\hat{n}, v > 5 \vee v < 5)$ replaces the predicate in the first line by $\mathbf{true} \wedge (v > 5 \vee v < 5)$
- $\mathtt{build}_{\mathtt{T}(\mathcal{G}_{\text{ex4.12}})}(\hat{n}, v < 5)$ replaces the predicate in the first line by $\mathbf{true} \wedge (v < 5)$.

Both solutions solve the TS problem, however the second one prevents the first branch to be ever taken. ◇

Given an assertion tree T and a branching node[5] $n \in T$ such that TS does not hold. One can invoke $\mathtt{build}_T(n, \psi)$ where ψ is either the disjunction of all the branching predicates or one of the branches predicate. If the function does not return $\bot$, then the TS problem is solved.

Recursion. The lifting algorithm can easily be extended to solve TS problems which occur in a recursive call, if we assume an annotation giving the invariant of its corresponding recursive definition (as in Definition 4.9). In fact, let a TS problem be located at a node $n \in T$ such that $\underline{n} = \chi\langle\vec{e}\rangle$ and let the invariant of the definition of χ be $\psi(\vec{v})$, then if the invocation of $\mathtt{build}_T(n, \psi\{\vec{e}/\vec{v}\})$ succeeds, the problem is solved.

[5] We also assume that TS is not violated in $\mathtt{parent}_T(n)\uparrow_T$ as in Definition 4.11.

In Example 4.13, below we give a more complex example of the application of Λ, with TS problems in recursive calls.

Example 4.13. Consider the global assertion below.

$$\begin{aligned}
\mathcal{G}_{\text{ex4.13}} = \;& \texttt{Generator} \to \texttt{Server}\colon \{n \mid n > 0\};\\
& \texttt{Player} \to \texttt{Server}\colon \{x \mid \mathbf{true}\};\\
& \mu\chi\langle x\rangle\{r \mid r > 0\}.\\
& \texttt{Server} \to \texttt{Player}\colon \Big\{ \{r > n\}\ \mathsf{less}: \quad \texttt{Player} \to \texttt{Server}\colon \{y \mid \mathbf{true}\}; \chi\langle y\rangle,\\
& \qquad\qquad \{r < n\}\ \mathsf{greater}: \ \texttt{Player} \to \texttt{Server}\colon \{z \mid \mathbf{true}\}; \chi\langle z\rangle,\\
& \qquad\qquad \{r = n\}\ \mathsf{win}: \quad \mathbf{0} \Big\}
\end{aligned}$$

modelling a small game where a `Player` has to guess an integer n, following the hints given by a `Server`. The number is fixed by a `Generator`. Each time `Player` sends `Server` a number, `Server` says whether n is less or greater than that number. Let $T_{\text{ex4.13}}$ be the tree generated from $\texttt{T}(\mathcal{G}_{\text{ex4.13}})$. There is a TS problem at the node corresponding to the recursive definition (let us call it n_3), indeed if $x \leqslant 0$, the invariant is not respected. After the first loop of $\Lambda(T_{\text{ex4.13}})$, the predicate $x > 0$ is added in the second interaction, i.e., $\texttt{TSres}_{T_{\text{ex4.13}}}(n_3)$ is invoked and returns a new tree, say $T'_{\text{ex4.13}}$, where the second interaction is updated to

$$\texttt{Player} \to \texttt{Server}\colon \{x \mid x > 0\}$$

Then, the algorithm loops two more times to solve the problems appearing before the recursive calls. Assuming n_4 (resp. n_5) is the node corresponding to the recursive call in the less (resp. greater) branch. $\texttt{TSres}_{T'_{\text{ex4.13}}}(n_4)$ is invoked, adding $y > 0$ in the interaction of the less branch, let us call this new tree $T''_{\text{ex4.13}}$. The updated interaction is now

$$\texttt{Player} \to \texttt{Server}\colon \{y \mid y > 0\}$$

Then, the algorithm invokes $\texttt{TSres}_{T''_{\text{ex4.13}}}(n_5)$, which adds $z > 0$ in the interaction of the greater branch, updating the interaction to

$$\texttt{Player} \rightarrow \texttt{Server}\colon \{z \mid z > 0\}$$

The global assertion now satisfies temporal satisfiability. ◇

Recursion parameters. Even though lifting may be applied when a TS violation is detected in a recursion definition, lifting a predicate involving a recursion parameter v would require to strengthen the invariant where v is introduced. This is quite dangerous, therefore the lifting algorithm does not apply in this case (cf. the last line of the first part of Definition 4.13). In fact, for recursive definition and calls, Definition 4.9 requires $\psi \implies \psi'\{\vec{e}/\vec{v}\}$, where ψ' is the recursion invariant and ψ is the conjunction of the previous predicates. Hence, lifting a predicate involving a recursion parameter may strengthen the invariant and possibly create a new problem in a corresponding recursive call. Moreover, notice that, in recursive calls, *GSat* (Definition 4.9) requires that $\psi \wedge \psi' \implies \psi'\{\vec{e}/\vec{v}\}$; namely, strengthening ψ' would automatically strengthen $\psi'\{\vec{e}/\vec{v}\}$ and therefore leave the TS problem unsolved.

We illustrate such a case with Example 4.14 below.

Example 4.14. Continuing with the example of Section 4.1, consider the following global

assertion which is supposed to have been "badly" modified by a designer.

$$
\begin{aligned}
\mathcal{G}_{\text{ex4.14}} \quad = \quad & \mathtt{b_i} \to \mathtt{s_i} : \{request \mid \mathbf{true}\}; && (1)\\
& \mathtt{s_i} \to \mathtt{b_i} : \{quote_f \mid \mathbf{true}\}; && (2)\\
& \mu\chi\langle quote_f\ quote_f\rangle\{quote_c\ quote_p \mid \mathtt{MIN} < quote_c \leqslant quote_p\}. && (3)\\
& \mathtt{b_i} \to \mathtt{s_i} : \ \{ \quad \{\mathbf{true}\}\ \mathsf{ok} : \quad \mathbf{0}, && (4)\\
& \qquad\qquad \{\mathbf{true}\}\ \mathsf{no} : \quad \mathtt{s_i} \to \mathtt{b_i} : \{quote_n \mid quote_n < quote_c\}; && (5)\\
& \qquad\qquad\qquad\qquad \chi\langle quote_n\ quote_c\rangle \ \} && (6)
\end{aligned}
$$

There are two TS problems in $\mathcal{G}_{\text{ex4.14}}$. The first problem is at the node corresponding to line (3) and is due to the fact that $quote_f$ is not required to be strictly greater than MIN – thus violate the invariant. The second problem is at the node corresponding to the recursive call (line (6)) and is due to the fact that $quote_n$ may not be strictly greater than MIN, and thus violate the invariant as well.

The first problem may be solved by lifting the predicate $\mathtt{MIN} < quote_f \leqslant quote_f$ to line (2), however our algorithm fails to solve the second problem as it would amount to lifting the predicate

$$\mathtt{MIN} < quote_n \leqslant quote_c$$

which contains a recursion parameter ($quote_c$).

We illustrate why lifting such a predicate is not supported by our algorithm. Assume that both problems were tackled by lifting, we would obtain the following global asser-

tion:

$$
\begin{aligned}
\mathcal{G}'_{\text{ex4.14}} \;=\;& \mathtt{b_i} \to \mathtt{s_i} : \{\mathit{request} \mid \mathbf{true}\}; && (1)\\
& \mathtt{s_i} \to \mathtt{b_i} : \{\mathit{quote}_f \mid \mathtt{MIN} < \mathit{quote}_f\}; && (2)\\
& \mu\chi\langle \mathit{quote}_f\ \mathit{quote}_f\rangle\{\mathit{quote}_c\ \mathit{quote}_p \mid \mathtt{MIN} < \mathit{quote}_c \leqslant \mathit{quote}_p\}. && (3)\\
& \mathtt{b_i} \to \mathtt{s_i} : \big\{\ \{\mathbf{true}\}\ \mathsf{ok} : \ \ \mathbf{0}, && (4)\\
& \qquad\qquad \{\mathbf{true}\}\ \mathsf{no} : \ \ \mathtt{s_i} \to \mathtt{b_i} : \{\mathit{quote}_n \mid \mathtt{MIN} < \mathit{quote}_n < \mathit{quote}_c\}; && (5)\\
& \qquad\qquad\qquad\qquad \chi\langle \mathit{quote}_n\ \mathit{quote}_c\rangle\ \big\} && (6)
\end{aligned}
$$

Observe that lifting $\mathtt{MIN} < \mathit{quote}_n \leqslant \mathit{quote}_c$ to line (3) yields the predicate:

$$\mathtt{MIN} < \mathit{quote}_c \leqslant \mathit{quote}_p \quad \wedge \quad \exists\, \mathit{quote}_n : \mathtt{MIN} < \mathit{quote}_n \leqslant \mathit{quote}_c$$

which is equivalent to the initial invariant.

In fact, there is also a TS problem in $\mathcal{G}'_{\text{ex4.14}}$. At line (5), if, e.g., $\mathit{quote}_f = \mathtt{MIN} + 1$, then $\mathtt{s_i}$ cannot choose a correct value for quote_n. Assuming again we may lift predicates that contain recursion parameters, we would lift the predicate $\mathtt{MIN} < \mathit{quote}_n < \mathit{quote}_c$ to solve this issue. This would strengthen the invariant to[6]

$$\mathtt{MIN} < \mathit{quote}_c \leqslant \mathit{quote}_p \quad \wedge \quad \exists\, \mathit{quote}_n : \mathtt{MIN} < \mathit{quote}_n < \mathit{quote}_c$$

which is equivalent to $\mathtt{MIN} + 1 < \mathit{quote}_c \leqslant \mathit{quote}_p$. At this point, we are essentially back to the global assertion $\mathcal{G}_{\text{ex4.14}}$, modulo the fact that MIN has increased by one. It is easy to see that, without the condition on recursion parameters in Definition 4.13, the algorithm would not terminate in such a case. ◇

[6] Observe the strict inequality between quote_n and quote_c in this case.

4.4.3 Properties of Λ

Similarly to the algorithms Σ and Π of Section 4.3, Λ does not modify the structure of the tree and preserves the properties of the initial assertion.

Proposition 4.6 (Underlying type structure - Λ)**.** *Let $\mathcal{G}$ be a global assertion. If $\Lambda(\mathcal{G})$ returns $\mathcal{G}'$ then $\texttt{erase}(\mathcal{G}) = \texttt{erase}(\mathcal{G}')$ (cf. Definition 4.8).*

Proof. The proof is by induction on the structure of $\mathcal{G}$, similarly to the one of Proposition 4.4. □

Proposition 4.7 below guarantees that Λ does not introduce new HS or TS problems. Likewise, it gives a formal account of the remark in Section 4.3.3, showing that Σ and Π do not add violations (of either HS or TS) to the amended global assertions.

Proposition 4.7 (Properties Preservation)**.** *Assume $F(\mathcal{G})$ returns $\mathcal{G}'$ with $F \in \{\Sigma, \Pi, \Lambda\}$. If $\overline{\texttt{HS}}(\mathcal{G}) = \varnothing$ then $\overline{\texttt{HS}}(\mathcal{G}') = \varnothing$ and if $\overline{\texttt{TS}}(\mathcal{G}) = \varnothing$ then $\overline{\texttt{TS}}(\mathcal{G}') = \varnothing$.*

Proof. We first consider HS preservation and then TS preservation for $F(\mathcal{G})$ with $F \in \{\Sigma, \Pi, \Lambda\}$.

HS preservation. The proof of HS preservation by Σ and Π follows by the fact that Σ and Π return $\mathcal{G}$ if $\overline{\texttt{HS}}(\mathcal{G}) = \varnothing$. For Λ, the preservation of HS follows from the fact that all the variables which are not known to a participant are quantified (either universally or existentially) in the modified predicates. We show that all the variables not known to the sender of an updated node are quantified. Let T be an assertion tree and ψ be the predicate lifted at a node $n \in T$ such that $\texttt{snd}_T(n) = \texttt{s}$. The predicate is quantified as in Definition 4.14 so to obtain $\forall \vec{x} : \exists \vec{y} : \psi$ such that

- $\vec{x} \subseteq \texttt{var}(\psi) \backslash \texttt{knows}_\texttt{s}(T)$ are introduced in a node in $n{\uparrow_T}$
- $\vec{y} \subseteq \texttt{var}(\psi)$ are introduced in a node in the subtree rooted at n

Let $z \in \mathtt{var}(\psi)$, (*i*) if $z \notin \vec{x}$, by definition, either z is known to $\mathtt{s}$ (therefore z should not be quantified) or z is introduced after n (hence it would have been quantified in $\vec{y}$). (*ii*) If $z \notin \vec{y}$, by definition, either z is introduced at n (therefore known to $\mathtt{s}$) or z is introduced before n. In that case, if it is known to $\mathtt{s}$ then it should not be quantified, and if it is not known to n, then $z \in \vec{x}$.

TS preservation. TS preservation in Σ follows from the fact that predicates may only be changed by a variable substitution. For $T = \mathtt{T}(\mathcal{G})$, such that $\overline{\mathtt{TS}}(\mathcal{G}) = \varnothing$, we have that, for any $n \in T$

$$\mathtt{PRED}_T(n) \implies \exists \mathtt{var}_T(n) : \phi$$

by definition of TS (Definition 4.9), with ϕ being the predicate at node n. And, by (4.3) (cf. page 126), we have that

$$\mathtt{PRED}_T(n) \implies \exists \mathtt{var}_T(n) : \phi\{^{v'}/_{v}\}$$

thus, TS is preserved by Σ.

TS preservation in Π follows from the fact that the predicates of a global assertion are only modified by adding equalities between problematic variables and fresh variables (see statement 2b in Proposition 4.5). For $T = \mathtt{T}(\mathcal{G})$, such that $\overline{\mathtt{TS}}(\mathcal{G}) = \varnothing$, we have that, for any $n \in T$

$$\mathtt{PRED}_T(n) \implies \exists \mathtt{var}_T(n) : \phi \tag{4.5}$$

by definition of TS, with ϕ being the predicate at node n. And, by construction of a $\prec_T$-chain, after each modification by Π, we obtain

$$\mathtt{PRED}_T(n) \wedge v = v_0 \wedge \ldots \wedge v_t = v_{t-1} \implies \exists \mathtt{var}_T(n) : \phi\{^{v_t}/_{v}\}$$

with $v_0 \ldots v_t$ fresh. This is equivalent to (4.5), i.e., TS is preserved by Π. The proof of TS preservation for Λ follows trivially from the first case of Definition 4.15. □

Proposition 4.8 establishes an intermediate result for the correctness of Λ. It says that a successful invocation of `TSres` (cf. Definition 4.14) on a node removes the problem at that node.

Proposition 4.8 (Correctness - `TSres`)**.** *Let T be an assertion tree. For each $n \in \overline{\mathtt{TS}}(T)$, if $\mathtt{TSres}_T(n) \neq \bot$, then $n \notin \overline{\mathtt{TS}}(\mathtt{TSres}_T(n))$.*

Proof. We start by giving the proof of the correctness for *interaction nodes*. Let T be an assertion tree with a node n such that $n \in \overline{\mathtt{TS}}(T)$, and

$$\underline{n} = \mathtt{s} \rightarrow \mathtt{r} \colon \{\vec{v} \mid \psi\}$$

with $\psi \iff \beta \wedge \gamma$ such that β *is in conflict* on $\mathtt{var}(n)$ with γ in $\mathtt{PRED}_T(n)$. Then β is the predicate to be lifted. Assume $\hat{T} = \mathtt{build}_T(n, \beta)$.

By Definition 4.13, we have that, for suitable $\vec{x}_1, \vec{y}_1 \ldots \vec{x}_k, \vec{y}_k$,

$$\mathtt{PRED}_{\hat{T}}(n) = \mathtt{PRED}_T(n) \wedge \forall \vec{x}_1 : \exists \vec{y}_1 : \beta \wedge \ldots \wedge \forall \vec{x}_k : \exists \vec{y}_k : \beta$$

We have that a quantified version of β is added k times in the assertion tree, above n. We show that

$$\bigwedge_{1 \leqslant i \leqslant k} \forall \vec{x}_i : \exists \vec{y}_i : \beta \implies \exists \vec{v} : \beta \tag{4.6}$$

Assume that each $\forall \vec{x}_i : \exists \vec{y}_i : \beta$ corresponds to the predicate added to the i^{th} node (n_i) modified by `TSres` (from the root to n). Then $\vec{y}_k = \vec{v} \cap \mathtt{var}(\beta)$ since by Definition 4.13 $\vec{y}_k$ is the set of variables introduced *after* n_k, and we assumed that the global assertion is closed (i.e., all the variables $\mathtt{var}(\beta)$ have been introduced before they are used, in n). Since every $\forall \vec{x}_i : \exists \vec{y}_i : \beta$ is satisfiable by Definition 4.13, we have that the following holds

$$\exists \vec{z} : \forall \vec{x}_k : \exists \vec{y}_k : \beta \quad \text{with } \vec{z} = \mathtt{var}(\forall \vec{x}_k : \exists \vec{y}_k : \beta)$$

this gives us (4.6) (note that $\vec{z}$, $\vec{x}_k$ and $\vec{y}_k$ are pairwise disjoint by definition). Since $\forall \vec{x}_k : \exists \vec{y}_k : \beta$ is one of the conjunct of $\texttt{PRED}_{\hat{T}}(n)$ we also have

$$\texttt{PRED}_{\hat{T}}(n) \implies \exists \vec{v} : \beta \tag{4.7}$$

By the definition of conflict (Definition 4.12), we have

$$\texttt{PRED}_T(n) \implies \exists \vec{v} : \gamma \quad \text{and} \quad \texttt{PRED}_T(n) \not\Longrightarrow \exists \vec{v} : \beta \wedge \gamma$$

hence, $\texttt{PRED}_T(n)$ is satisfiable. Therefore, by weakening, we have that:

$$\texttt{PRED}_{\hat{T}}(n) \implies \exists \vec{v} : \gamma \tag{4.8}$$

TS must hold for n, which implies that $n \notin \overline{\texttt{TS}}(\hat{T})$ and $\texttt{TSnode}_{\hat{T}}(n)$ holds, i.e.,

$$\texttt{PRED}_{\hat{T}}(n) \implies \exists \vec{v} : \psi \quad (\text{with } \psi \iff \beta \wedge \gamma)$$

Otherwise, that would imply that

$$\texttt{PRED}_{\hat{T}}(n) \wedge \forall \vec{v} : (\neg\beta \vee \neg\gamma)$$

which is in contradiction with (4.7) (β) and (4.8) (γ).

Let us now show the result for *recursive nodes*, which is somewhat similar to the previous case. Assume we haveg

$$\underline{n} = \mu\chi\langle\vec{e}\rangle\{\vec{v} \mid \beta\}$$

with $n \in \overline{\mathtt{TS}}(T)$, thus we have that (by Definition 4.9)

$$\mathtt{PRED}_T(n) \Longrightarrow \beta\{\vec{e}/\vec{v}\} \tag{4.9}$$

Assuming $\hat{T} = \mathtt{build}_T(n, \beta\{\vec{e}/\vec{v}\})$ (i.e., `build` succeeds), we have that a quantified version of $\beta\{\vec{e}/\vec{v}\}$ is added k times in the assertion tree, above n. Following a similar argument as before, this gives us

$$\mathtt{PRED}_{\hat{T}}(n) = \mathtt{PRED}_T(n) \wedge \beta\{\vec{e}/\vec{v}\}$$

By Definition 4.13, we also know that $\beta\{\vec{e}/\vec{v}\}$ is satisfiable. Moreover, $\mathtt{PRED}_T(n)$ is satisfiable, by Definition 4.11 and (4.9).

Since

$$\mathtt{PRED}_{\hat{T}}(n) \Longrightarrow \beta\{\vec{e}/\vec{v}\} \Longleftrightarrow \mathtt{PRED}_T(n) \wedge \beta\{\vec{e}/\vec{v}\} \Longrightarrow \beta\{\vec{e}/\vec{v}\}$$

we have that $n \notin \overline{\mathtt{TS}}(\hat{T})$. □

Finally, we can say that, if a repeated application of lifting succeeds, the global assertion which is returned satisfies temporal satisfiability.

Theorem 4.2 (Correctness - Λ)**.** *If* $\Lambda(\mathcal{G}) = \mathcal{G}'$ *then* $\overline{\mathtt{TS}}(\mathcal{G}') = \varnothing$.

Proof. The proof is by induction on the number of problematic nodes and the minimum depth of these nodes in the tree. It relies on Proposition 4.8, i.e., the fact that $\mathtt{TSres}_T(n)$ either solves the problem at n or fails.

Let $T = \mathtt{T}(\mathcal{G})$ and N be the set of nodes in T which violates TS. We write $|n|$ for the depth of n in T (with $|T^\bullet| = 0$), and we denote by N' the number of problematic node after an invocation to `TSres`.

1. If $N = \varnothing$, then T is TS.

2. If $N \neq \varnothing$, let $n \in \overline{\texttt{TS}}(T) \subseteq N$, after an invocation to $\texttt{TSres}_T(n)$, we have

   (a) If $|n| > 1$ then either

      i. $N' := N \backslash \{n\}$, i.e., the node is simply removed from the set of problematic nodes,

      ii. $N' := N \cup N_* \backslash \{n\}$, where N_* is the set of problematic nodes *added* by `TSres`. We have that $\forall n'_i \in N_* : \ |n'_i| < |n|$, i.e., the problem at n is solved but other problematic nodes, *above* n in T, are added, or,

      iii. the algorithm fails on n

   (b) If $|n| \leqslant 1$ then either $N' := N \backslash \{n\}$, or the algorithm fails. In fact, once the algorithm reaches a problem located at a child of the root, then it either fails or solves the problem. Indeed, there cannot be a TS problem at the root node unless the predicate is unsatisfiable (see Definition 4.9), in which case, the algorithm fails.

   Note that selecting $n \in \overline{\texttt{TS}}(T)$ implies that the depth of n is smaller or equal to the depth of the nodes in N.

Regarding step 2(a)ii, note that the algorithm cannot loop on a problematic node indefinitely. Indeed, the number of (sub)predicates available for lifting is finite and Λ invokes `TSres` only when a problematic node is found.

It is easily shown by induction that the algorithm terminates either with $\overline{\texttt{TS}}(T) = \varnothing$, or a failure. □

In addition, we have that Λ preserves the domain of possible values for each variable from the initial assertion.

Proposition 4.9 (Assertion predicates)**.** *If $\Lambda(\mathcal{G}) = \mathcal{G}'$ then for all $n \in \texttt{T}(\mathcal{G})$ such that n is*

a leaf, and its corresponding node $n' \in \mathtt{T}(\mathcal{G}')$

$$\mathtt{PRED}_{\mathtt{T}(\mathcal{G})}(n) \wedge \mathtt{cst}_{\mathtt{T}(\mathcal{G})}(n) \iff \mathtt{PRED}_{\mathtt{T}(\mathcal{G}')}(n') \wedge \mathtt{cst}_{\mathtt{T}(\mathcal{G}')}(n')$$

Proof. The proof follows from the observation that predicates are only duplicated in the tree, i.e., the lifting algorithm does not add any new constraints in the conjunction of the predicates found on the path from the root to a leaf. In addition, the algorithm modifies only predicates which appear above a problematic node in the tree (i.e., a predicate in a leaf will never be modified, by Definition 4.13).

We show the result for *interaction nodes*, the case for recursive nodes is similar. After each successful iteration of Λ on an assertion tree $T = \mathtt{T}(\mathcal{G})$, where

$$\mathtt{PRED}_T(n) = \psi_1 \wedge \ldots \wedge \psi_i \wedge \ldots \wedge \psi_k$$

and ψ_i is the predicate where the TS problem is located, and β is the lifted predicate (cf. proof of Proposition 4.8), such that we have, by Definition 4.12

$$\psi_i \iff \beta \wedge \gamma$$

We can rewrite $\mathtt{PRED}_T(n)$ as

$$\mathtt{PRED}_T(n) = \psi_1 \wedge \ldots \wedge \gamma \wedge \beta \wedge \ldots \wedge \psi_k$$

`build` returns a new tree T' such that the conjunction of predicates in the new tree is of the form

$$\mathtt{PRED}_{T'}(n') = \mathtt{PRED}_T(n) \wedge \bigwedge_{j \in J} \forall \vec{x_j} \exists \vec{y_j} : \beta$$

By Definition 4.13, we have $\vec{x}_j\,\vec{y}_j \subseteq \mathtt{var}(\beta)$, for all $j \in J$. Therefore, we have

$$\beta \implies \forall \vec{x}_j \exists \vec{y}_j : \beta \quad \text{for all } j \in J$$

This means that the additional predicates do not constrain further the conjunction of predicates, and we have

$$\mathtt{PRED}_T(n) \iff \mathtt{PRED}_{T'}(n') \qquad \square$$

4.5 A Methodology for Amending Choreographies

The algorithms Σ, Π, and Λ of Section 4.3 and Section 4.4 can be used to support a methodology for amending contracts in choreographies. The methodology consists of the following steps:

- (i) the architect designs a global assertion $\widehat{\mathcal{G}}$ (possibly based on a synthesised global type synthesised as in Chapter 3)
- (ii) the architect is notified if there are any HS or TS problems in $\widehat{\mathcal{G}}$
- (iii) using Σ and Π solutions may be offered for HS problems, while Λ can be used to offer solutions and/or hints on how to solve TS problems
- (iv) the architect selects one of the solutions offered in (iii)
- (v) steps (ii) to (v) are repeated until all problems are addressed.

We illustrate our methodology using the following global assertion:

$$
\begin{aligned}
\widehat{\mathcal{G}} \;=\;& \mu\chi\langle 10\rangle\{v \mid v > 0\}. && (1)\\
& \texttt{Alice}\rightarrow\texttt{Bob}\colon \{v_1 \mid v \geqslant v_1\}; && (2)\\
& \texttt{Bob}\rightarrow\texttt{Carol}\colon \{v_2 \mid v_2 > v_1\}; && (3)\\
& \texttt{Carol}\rightarrow\texttt{Alice}\colon \{v_3 \mid v_3 > v_1\}; && (4)\\
& \texttt{Carol}\rightarrow\texttt{Bob}\colon \{v_4 \mid v_4 > v\}; && (5)\\
& \texttt{Alice}\rightarrow\texttt{Bob}\colon \;\Big\{\; \{\mathbf{true}\}\,\mathsf{cont}\colon \chi\langle v_1\rangle, && (6)\\
& \qquad\qquad \{\mathbf{true}\}\,\mathsf{finish}\colon \texttt{Alice}\rightarrow\texttt{Bob}\colon \{v_5 \mid v_1 < v_5 < v_3 - 2\} \;\Big\} && (7)
\end{aligned}
$$

which extends the global assertion in Example 4.4 and is assumed to have been designed by the architect (step (i) of the methodology).

Firstly, $\widehat{\mathcal{G}}$ is inspected by history sensitivity and temporal satisfiability checkers, such as the ones described in [52]. If any HS problems are reported (step (ii) of the methodology), algorithms Σ and Π are used, while Λ is used for TS problems. This allows the architect to detect all the problems and to consider the ones for which (at least) one of the algorithms is applicable. In general, the architect can decide which problem to tackle first (step (iii) of our methodology). For $\widehat{\mathcal{G}}$, we focus on HS problems first. There are two HS problems in $\widehat{\mathcal{G}}$, both of which can be solved automatically, and the methodology will return the following suggestions.

1. At line (4), v_1 is not known by `Carol`; the problem is solvable by either

   - replacing $v_3 > v_1$ by $v_3 > v_2$ (algorithm Σ) at line (4), or
   - by revealing v_1 to `Carol` (algorithm Π); in this case, line (3) becomes

     $$\texttt{Bob}\rightarrow\texttt{Carol}\colon \{v_2\, u_1 \mid v_2 > v_1 \wedge u_1 = v_1\}$$

     and the predicate at line (4) becomes $v_3 > u_1$.

2. At line (5), v is not known by `Carol`; the problem is solvable by revealing the value of v to `Carol` (algorithm Π) in which case line (3) becomes

$$\texttt{Bob} \to \texttt{Carol}\colon \{v_2\ u_2 \mid v_2 > v_1 \wedge u_2 = v\}$$

and the assertion at line (5) becomes $v_4 > u_2$.

In the *propagation case* (i.e., Π), the methodology gives the architect information on which participants the value of a variable may be disclosed to. Indeed, as discussed in Remark 4.5, it may not be appropriate to use the suggested solution. Therefore, the actual adoption of the proposed solutions should be left to the architect. In addition, the order in which problems are tackled is also left to the architect (e.g., the same variable may be involved in several problems and solving one of them may automatically fix the others). Assuming that Σ is used to solve the first problem and Π to solve the second, HS is fixed, and the amended global assertion is as follows:

$$\begin{aligned}
\widehat{\mathcal{G}} \;=\;& \mu\chi\langle 10\rangle\{v \mid v > 0\}. && (1)\\
& \texttt{Alice} \to \texttt{Bob}\colon \{v_1 \mid v \geqslant v_1\}; && (2)\\
& \texttt{Bob} \to \texttt{Carol}\colon \{v_2\ u_1 \mid v_2 > v_1 \wedge u_1 = v\}; && (3)\\
& \texttt{Carol} \to \texttt{Alice}\colon \{v_3 \mid v_3 > v_2\}; && (4)\\
& \texttt{Carol} \to \texttt{Bob}\colon \{v_4 \mid v_4 > u_1\}; && (5)\\
& \texttt{Alice} \to \texttt{Bob}\colon \ \Big\{ \ \{\mathbf{true}\}\ \mathsf{cont} : \chi\langle v_1\rangle, && (6)\\
& \qquad\qquad \{\mathbf{true}\}\ \mathsf{finish} : \texttt{Alice} \to \texttt{Bob}\colon \{v_5 \mid v_1 < v_5 < v_3 - 2\}\ \Big\} && (7)
\end{aligned}$$

Now HS is satisfied in $\widehat{\mathcal{G}}$, but TS problems are still present.

In case a TS problem cannot be solved automatically, additional information can be returned: (a) at which node the problem occurred, (b) which variables or recursion parameters are posing problems (i.e., using `split` and `build`), and (c) where liftings are not possible (i.e., when `build` fails to add a satisfiable predicate to a node). For $\widehat{\mathcal{G}}$ there

are two TS problems which are dealt with sequentially. The methodology would report that

1. At line (6), v_1 does not satisfy the invariant $v > 0$. This can be solved by lifting $v_1 > 0$ (i.e., the invariant where v is replaced by the actual parameter v_1) to the interaction at line (2), which would yield the new predicate $v \geqslant v_1 \wedge v_1 > 0$.

2. At line (7), there might be no value for v_5 such that $v_1 < v_5 < v_3 - 2$. The assertion is *in conflict* (cf. Definition 4.12) with the previous predicates; this problem cannot be solved since lifting would add the following predicates in lines (2) and (4), respectively.

   - $\exists\, v_3\, v_5 :\ v_1 < v_5 < v_3 - 2$ which is indeed satisfiable, but remarkably does not constraint v_1 more than the initial predicate. Indeed, the updated predicate (i.e., $v \geqslant v_1 \wedge \exists\, v_3\, v_5 :\ v_1 < v_5 < v_3 - 2$) does not constrain v_1 more than the original predicate, $v \geqslant v_1$.
   - $\forall\, v_1 :\ \exists\, v_5 :\ v_1 < v_5 < v_3 - 2$ which is not satisfiable, therefore the algorithm fails.

The failure of Λ is due to the fact that v_5 is constrained by v_1 and v_3 which are fixed by two different participants. They would have to somehow interact in order to guarantee that there exists a value for v_5, this cannot be done using the proposed algorithms. Notice that in this case the methodology tells the architect that v_5, fixed by `Alice`, is constrained by v_1 and v_3 which are fixed by `Alice` and `Carol`, respectively. Our approach can also suggest that the node introducing v_3, or (the part of) the assertion over v_3 may be the source of the problem since v_3 is the only variable not known by `Alice`.

4.5.1 Amendment Strategies

The methodology above does not specify any particular order for tackling HS and TS problems. In fact, it is for the architect to assess the importance of each problem; furthermore, the architect should also proof-read the proposed solutions (e.g., in the case of propagation). One of our future work plans is to help the architect making choice regarding the order in which problems should be tackled by designing *amendment strategies*, which maximise the chances of having *all* problems solved using the proposed algorithms.

In fact, the application of an algorithm could be spoilt by the application of another one. For instance, the application of the strengthening algorithm (Σ) might compromise the applicability of the lifting algorithm (Λ), and vice versa. This happens when a variable v introduced by Σ in a node (say n) to solve an HS problem is also involved in a TS problem, in a descendant node (say n') of n; indeed, both Σ and Λ will independently strengthen the predicate of n. This may compromise the application of the algorithm invoked last, as illustrated below.

Let T be an assertion tree, where there are $n, n' \in T$ such that $\texttt{varHS}_T(n) \neq \varnothing$, n' is a descendant of n,

$$\underline{n} = \mathtt{s} \to \mathtt{r} : \{\vec{v}_1\ v\ \vec{v}_2 \mid \psi\} \qquad \text{and} \qquad \underline{n'} = \mathtt{s}' \to \mathtt{r}' : \{\vec{u} \mid \gamma \wedge \beta\}$$

where $v \in \texttt{var}(\beta)$, and β is in conflict on $\vec{u}$ with γ in $\texttt{PRED}_T(n')$.

- if Σ is used to solve the problem at n, ψ might be strengthened (by a variable substitution)
- if Λ is used to solve the problem at n', β will be lifted to the node n since $v \in \texttt{var}(\beta)$.

Call ψ_1 the new predicate produced by Σ and ψ_2 the one produced by Λ. The application of Σ would give

$$\underline{n} = \mathtt{s} \to \mathtt{r} : \{\vec{v}_1\ v\ \vec{v}_2 \mid \psi_1\}$$

which might prevent the application of Λ because, e.g., $\psi_1 \wedge \forall \vec{x} : \exists \vec{y} : \beta$ is not satisfiable, for suitable $\vec{x}$ and $\vec{y}$. Likewise, the application of Λ first would give

$$\underline{n} = \mathtt{s} \rightarrow \mathtt{r} : \{\vec{v}_1 \; v \; \vec{v}_2 \mid \psi_2\}$$

for which Σ might not be applicable because, e.g., no substitution with a variable known to $\mathtt{s}$ yields a satisfiable predicate for $\underline{n}$.

We conjecture that this is the *only* source of issues arising from the absence of prescribed order for addressing HS and TS problems in the methodology. Intuitively, the only way one algorithm could spoil the applicability of another is by modifying the satisfiability of a predicate of (at least) one common node. Propagation (Π) preserves the semantics of all the nodes it updates (by Proposition 4.5); instead, Σ and Λ may strengthen predicates. Note that Σ modifies only the node in which there is an HS problem, while Λ updates only the nodes *above* the TS-problematic one. Therefore, the only possible issue occurs when there is a node with an HS problem "above" another, with a TS problem. Since Λ modifies only the nodes that introduce variables which appear in a *problematic* predicate, we conjecture that this happens only in the case explained above.

Note that even though an occurrence of two inter-dependent TS and HS problems as above may compromise the applicability of an algorithm, thus preventing the amendment of existing problems, it will not introduce new violations, as stated in Proposition 4.7.

4.6 Applying the Methodology

To illustrate our methodology, we consider the design of a couple of services offered by an ATM to the customers of the bank where it is located. The first service offers cash withdrawal. The second service allows customers to request a small line of credit, provided that they are considered trusted by the bank. We propose two global assertions

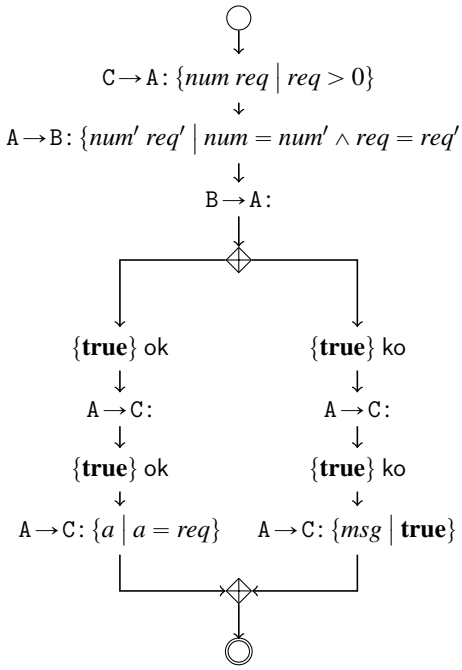


Figure 4.2: ATM protocol

for each of the two functionalities and discuss problems which may be encountered during their design.

4.6.1 Cash Withdrawal

Consider the following global assertion, also illustrated in Figure 4.2, where C is the customer, A is the ATM, and B is the bank.

$$\mathtt{C} \to \mathtt{A}\colon \{num\ req \mid req > 0\}; \tag{1}$$

$$\mathtt{A} \to \mathtt{B}\colon \{num'\ req' \mid num = num' \wedge req = req'\}; \tag{2}$$

$$\mathtt{B} \to \mathtt{A}\colon \Big\{ \quad \{\mathbf{true}\}\ \mathsf{ok} : \mathtt{A} \to \mathtt{C}\colon \{\mathbf{true}\}\ \mathsf{ok} : \mathtt{A} \to \mathtt{C}\colon \{a \mid a = req\}, \tag{3}$$

$$\{\mathbf{true}\}\ \mathsf{ko} : \mathtt{A} \to \mathtt{C}\colon \{\mathbf{true}\}\ \mathsf{ko} : \mathtt{A} \to \mathtt{C}\colon \{msg \mid \mathbf{true}\}\ \Big\} \tag{4}$$

This global assertion models a simple cash withdrawal service under the assumption that the credentials of the customer have already been verified. The customer sends the ATM an account number *num* and the amount of money to withdraw *req*. The ATM

forwards the request to the bank. If the withdrawal is accepted, the bank selects branch ok; in this case the ATM gives the corresponding amount to the customer. Otherwise, the bank selects branch ko and ATM sends an error message to the customer.

This global assertion is well-asserted, but soon the architect realises that it contains a major flaw: the ATM is expected to give money to the customer even when there is not enough cash available in the machine. The architect corrects the problem by adding a predicate $a \leqslant \texttt{CASH}$ at line (3), where CASH is the money available at the beginning of the session:

$$\texttt{C} \to \texttt{A}\colon \{num\ req \mid req > 0\}; \tag{1}$$

$$\texttt{A} \to \texttt{B}\colon \{num'\ req' \mid num = num' \wedge req = req'\}; \tag{2}$$

$$\begin{aligned} \texttt{B} \to \texttt{A}\colon \ \Big\{ \ & \{\mathbf{true}\}\ \mathsf{ok} : \texttt{A} \to \texttt{C}\colon \{\mathbf{true}\}\ \mathsf{ok} : \texttt{A} \to \texttt{C}\colon \{a \mid a = req \wedge\ a \leqslant \texttt{CASH}\}, && (3)\\ & \{\mathbf{true}\}\ \mathsf{ko} : \texttt{A} \to \texttt{C}\colon \{\mathbf{true}\}\ \mathsf{ko} : \texttt{A} \to \texttt{C}\colon \{msg \mid \mathbf{true}\}\ \Big\} && (4) \end{aligned}$$

Although this solves the flaw, a temporal satisfiability issue is introduced at line (3). In fact, A cannot guarantee its obligation if the amount requested *req* in the first interaction is greater than the cash available.

Fortunately, Λ is applicable and it can amend the global assertion automatically by returning the choreography below

$$\texttt{C} \to \texttt{A}\colon \{num\ req \mid req > 0 \wedge \exists a :\ a = req \wedge\ a \leqslant \texttt{CASH}\}; \tag{1}$$

$$\texttt{A} \to \texttt{B}\colon \{num'\ req' \mid num = num' \wedge req = req'\}; \tag{2}$$

$$\begin{aligned} \texttt{B} \to \texttt{A}\colon \ \Big\{ \ & \{\mathbf{true}\}\ \mathsf{ok} : \texttt{A} \to \texttt{C}\colon \{\mathbf{true}\}\ \mathsf{ok} : \texttt{A} \to \texttt{C}\colon \{a \mid a = req \wedge\ a \leqslant \texttt{CASH}\}, && (3)\\ & \{\mathbf{true}\}\ \mathsf{ko} : \texttt{A} \to \texttt{C}\colon \{\mathbf{true}\}\ \mathsf{ko} : \texttt{A} \to \texttt{C}\colon \{msg \mid \mathbf{true}\}\ \Big\} && (4) \end{aligned}$$

which is well-asserted.

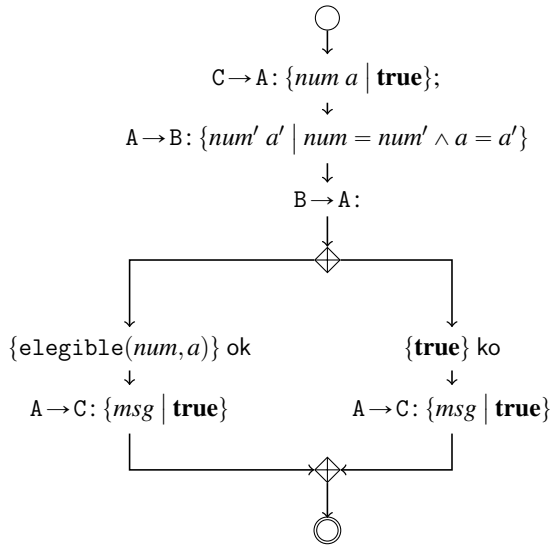


Figure 4.3: Credit request protocol

4.6.2 Credit Request

We now want to model a service through which a customer can request a small line of credit. The intuition of the protocol is illustrated in Figure 4.3. The customer C sends their account number *num* and the requested credit *a* to the ATM. The ATM forwards the request to the bank and, depending on whether or not C is eligible for the credit according to the bank's records (i.e., $\texttt{eligible}(num, a)$), the bank selects either branch ok or ko. Finally, the ATM sends a message to the customer notifying them of the decision.

Remark 4.10. *For simplicity, we use* branch mergeability *[73], a slight extension of multiparty session types. Otherwise, it would be necessary to add an extra branch in the inner branching between* A *and* C *to have the same behaviour of* C *in both branches of the outer branching. Note that this does not affect the applicability of our methodology.*

A naive global assertion modelling this service is as follows:

$$\begin{aligned}
&\mathtt{C} \to \mathtt{A}\colon \{num\ a \mid \mathbf{true}\}; && (1)\\
&\mathtt{A} \to \mathtt{B}\colon \{num'\ a' \mid num = num' \wedge a = a'\}; && (2)\\
&\mathtt{B} \to \mathtt{A}\colon \Big\{ \quad \{\mathtt{eligible}(num, a)\} \quad \mathsf{ok} : \mathtt{A} \to \mathtt{C}\colon \{msg \mid \mathbf{true}\}, && (3)\\
&\qquad\qquad \{\mathbf{true}\} \qquad\qquad\quad \mathsf{ko} : \mathtt{A} \to \mathtt{C}\colon \{msg \mid \mathbf{true}\}\Big\} && (4)
\end{aligned}$$

The attentive reader will notice that there is an HS problem at line (3) of this global assertion. Indeed, B does not know *num* nor a and therefore could not guarantee that the customer is in fact eligible. Both Σ and Π algorithms are applicable here. The second algorithm would return the following global assertion:

$$\begin{aligned}
&\mathtt{C} \to \mathtt{A}\colon \{num\ a \mid \mathbf{true}\}; && (1)\\
&\mathtt{A} \to \mathtt{B}\colon \{num'\ a'\ v_1\ v_2 \mid num = num' \wedge a = a' \wedge v_1 = num \wedge v_2 = a\}; && (2)\\
&\mathtt{B} \to \mathtt{A}\colon \Big\{ \quad \{\mathtt{eligible}(v_1, v_2)\} \quad \mathsf{ok} : \mathtt{A} \to \mathtt{C}\colon \{msg \mid \mathbf{true}\}, && (3)\\
&\qquad\qquad \{\mathbf{true}\} \qquad\qquad\quad \mathsf{ko} : \mathtt{A} \to \mathtt{C}\colon \{msg \mid \mathbf{true}\}\Big\} && (4)
\end{aligned}$$

Although this solves the problem, we notice that this solution is not ideal. Indeed v_1 and v_2 are somewhat redundant with num' and a', respectively.

Strengthening gives us a better solution in this case:

$$\begin{aligned}
&\mathtt{C} \to \mathtt{A}\colon \{num\ a \mid \mathbf{true}\}; && (1)\\
&\mathtt{A} \to \mathtt{B}\colon \{num'\ a' \mid num = num' \wedge a = a'\}; && (2)\\
&\mathtt{B} \to \mathtt{A}\colon \Big\{ \quad \{\mathtt{eligible}(num', a')\} \quad \mathsf{ok} : \mathtt{A} \to \mathtt{C}\colon \{msg \mid \mathbf{true}\}, && (3)\\
&\qquad\qquad \{\mathbf{true}\} \qquad\qquad\quad \mathsf{ko} : \mathtt{A} \to \mathtt{C}\colon \{msg \mid \mathbf{true}\}\Big\} && (4)
\end{aligned}$$

Which is what one would expect. Note that the algorithm is applicable because

$$num = num' \wedge a = a' \wedge \mathtt{eligible}(num', a') \implies \mathtt{eligible}(num, a)$$

holds and B knows num' and a'.

4.7 Concluding Remarks

In this chapter, we investigated the problem of designing consistent assertions. We focused on two consistency criteria from [12]: history sensitivity and temporal satisfiability. We proposed and compared three algorithms (Σ, Π, and Λ) to amend global assertions. Since each algorithm is applicable only in certain circumstances, we proposed a methodology that supports the architect when violations are not automatically amendable.

On the theoretical side, the algorithms Σ, Π, and Λ address the general problem of guaranteeing the satisfiability of predicates when: (1) the parts of the system have a different perspective/knowledge of the available information (in the case of history sensitivity), and (2) the constraints are introduced progressively (in the case of temporal satisfiability). The proposed solutions can be adapted and used, for instance, to amend processes (rather than types), orchestrations (rather than choreographies, when we want to check for local constraints) expressed in formalisms such as CC-Pi [18], a language for distributed processes with constraints. Interestingly, temporal satisfiability is similar to the feasibility property in [2] requiring that any initial segment of a computation must be possibly extended to a full computation to prevent "a scheduler from 'painting itself into a corner' with no possible continuation".

An interesting future development is to investigate more general accounts of satisfiability which is applicable to different scenarios. In scope of future work, we will design *amendment strategies* to so to refine our methodology and maximise the applicability of the proposed algorithms (cf. Section 4.5.1).

CHAPTER 5

Choreography Synthesis as Contract Agreement

We introduce a formal model for distributed systems where each participant advertises its requirements and obligations as behavioural *contracts* (in the form of local types), and where multiparty sessions are started when a set of contracts allows to synthesise a *choreography* (i.e., a global type). The framework is based on the CO_2 calculus and preserves its main features, while allowing to establish sessions where the number of participants is not determined beforehand. We give progress and session fidelity results for CO_2 systems which are "honest".

5.1 Introduction

Distributed applications are nowadays omnipresent but even for seemingly simple cases, there is still a pressing need to make sure they do work as their designers intended. Indeed, developing and maintaining such systems is rather difficult. This is not only due to intrinsic issues originating from concurrency and distribution; but also because such applications have to be designed within a strange paradox: they are made of components that, on the one hand, must collaborate with each other and, on the other hand, may compete for resources, or for achieving conflicting goals.

We propose a formal model for distributed systems based on *contracts-driven interac-*

tions: components advertise contracts (in the form of local types); such contracts are used at runtime to establish (multiparty) *agreements*, and such agreements steer the behaviour of components. Therefore, contracts are not just a specification or a design mechanism anymore, rather they become a pivotal element of the execution model.

We combine two approaches: *session types* [44] and *contract-oriented computing* [8]. From the former, we adopt concepts, syntax, and semantics – and in particular, the interplay between local behaviours and choreographies (i.e., between local types and global types) as a method for specifying and analysing the interactions of participants in a distributed system. However, in our framework we do not assume that a participant will necessarily always adhere to its specification, nor that a global description is available beforehand to validate the system. From the contract-oriented computing approach, we adopt CO_2 [7] (for COntract-Oriented computing), a generic contract-oriented calculus where participants advertise their requirements and obligations through contracts, and interact with each other once *compliant* contracts have been found. In this work, we tailor CO_2 to a multiparty model where contracts have the syntax of local types. We say that contracts $P_1, \ldots, P_n$ are compliant when they can be used to synthesise a global type (cf. Chapter 3). Once a set of compliant contracts has been found, a CO_2 session may be established, wherein the participants who advertised the contracts can interact. However, in line with what may happen in real life cases, the runtime behaviour of these participants may then depart from the contracts: the calculus allows to model and identify these situations.

Our framework models multiparty contractual agreements as "tangible" objects, i.e., choreographies. This allows us to rely on results from Chapter 3; in particular, the fact that well-formed choreographies ensure that contractual agreements enjoy safety and liveness properties. Furthermore, it allows us to easily check that some meta-level properties are satisfied at runtime, e.g., on the number of involved participants, whether or not the session may terminate, etc.

Our adaptation of CO_2 to a choreography-based contract model preserves the properties of the original calculus. In particular, if a system gets stuck, it is possible to identify which participants violated their contracts.

We also introduce global progress and session fidelity in CO_2, inspired by analogous concepts in theories based on session types (cf. Section 2.1.2). We show that they hold in systems where all participants are *honest* (i.e., they always respect their contracts in any context) – even when a participant takes part in multiple sessions.

5.2 A Motivating Example

We introduce a running example to illustrate our framework. We denote participant variables in bold font, i.e., $\mathbf{s}$, $\mathbf{b_1}$, $\mathbf{b_2}$, ... are participant variables; while $\mathtt{s}$, $\mathtt{b}_1$, $\mathtt{b}_2$, ... are participant identifiers, as in previous chapters.

Consider the following distributed scenario: an online store $\mathtt{s}$ allows any two buyers $\mathbf{b_1}$ and $\mathbf{b_2}$ to make a joint purchase through a simplified protocol: once they both have requested the same item, a quote is sent to $\mathbf{b}_1$, who is then expected to either place an order (order) or end the session (bye); the store also promises to let $\mathbf{b}_2$ know whether the order was placed (ok) or cancelled (bye). The behaviour of the store $\mathtt{s}$ is described by the following contract:

$$P_{\mathtt{s}} \;=\; \mathbf{b_1}?\mathsf{req}\,;\mathbf{b_2}?\mathsf{req}\,;\mathbf{b_1}!\mathsf{quote}\,;(\mathbf{b_1}?\mathsf{order}\,;\mathbf{b_2}!\mathsf{ok} \;+\; \mathbf{b_1}?\mathsf{bye}\,;\mathbf{b_2}!\mathsf{bye})$$

We would like to know which contracts would be *compliant* with $P_{\mathtt{s}}$. A possible pair of compliant contracts, advertised by buyers $\mathtt{b}_1$ and $\mathtt{b}_2$, is as follows:

$$\begin{aligned} P_{\mathtt{b}_1} &= \mathbf{s}!\mathsf{req}\,;\mathbf{s}?\mathsf{quote}\,;\big(\mathbf{b_2'}!\mathsf{ok}\,;\mathbf{s}!\mathsf{order} \oplus \mathbf{b_2'}!\mathsf{bye}\,;\mathbf{s}!\mathsf{bye}\big) \\ P_{\mathtt{b}_2} &= \mathbf{s'}!\mathsf{req}\,;\big(\mathbf{b_1'}?\mathsf{ok}\,;\mathbf{s'}?\mathsf{ok} \;+\; \mathbf{b_1'}?\mathsf{bye}\,;\mathbf{s'}?\mathsf{bye}\big) \end{aligned}$$

where $\mathtt{b}_1$ promises to send the request to the store ($\mathbf{s}$), wait for the quote, and then notify the other buyer ($\mathbf{b}_2'$) before accepting or rejecting the store offer. Symmetrically, $\mathtt{b}_2$ promises to sends the request to the store ($\mathbf{s}'$), and then expects to receive the same notification (either ok or bye) from both the other buyer ($\mathbf{b}_1'$) and the store $\mathbf{s}'$.

An agreement among $\mathtt{s}$, $\mathtt{b}_1$ and $\mathtt{b}_2$ may be found by replacing the participant variables in each contract with actual identifiers, and composing them in a system such as:

$$S_{\mathtt{s}\mathtt{b}_1\mathtt{b}_2} \;=\; \mathtt{s}[P_{\mathtt{s}}\,\{{}^{\mathtt{b}_1}/_{\mathbf{b_1}}, {}^{\mathtt{b}_2}/_{\mathbf{b_2}}\}] \;\mid\; \mathtt{b}_1[P_{\mathtt{b}_1}\,\{{}^{\mathtt{s}}/_{\mathbf{s}}, {}^{\mathtt{b}_2}/_{\mathbf{b}_2'}\}] \;\mid\; \mathtt{b}_2[P_{\mathtt{b}_2}\,\{{}^{\mathtt{s}}/_{\mathbf{s}'}, {}^{\mathtt{b}_1}/_{\mathbf{b}_1'}\}]$$

from which we may obtain the following global type, as per the construction of Chapter 3,

$$\begin{aligned} \mathcal{G}_{\mathtt{s}\mathtt{b}_1\mathtt{b}_2} \;=\;& \mathtt{b}_1 \to \mathtt{s} : \mathsf{req};\ \mathtt{b}_2 \to \mathtt{s} : \mathsf{req};\ \mathtt{s} \to \mathtt{b}_1 : \mathsf{quote}; \\ & \left(\begin{array}{c} \mathtt{b}_1 \to \mathtt{b}_2 : \mathsf{ok};\ \mathtt{b}_1 \to \mathtt{s} : \mathsf{order};\ \mathtt{s} \to \mathtt{b}_2 : \mathsf{ok} \\ + \\ \mathtt{b}_1 \to \mathtt{b}_2 : \mathsf{bye};\ \mathtt{b}_1 \to \mathtt{s} : \mathsf{bye};\ \mathtt{s} \to \mathtt{b}_2 : \mathsf{bye} \end{array} \right) \end{aligned}$$

Recall that the synthesis of $\mathcal{G}_{\mathtt{s}\mathtt{b}_1\mathtt{b}_2}$ guarantees that the global type is well-formed and projectable, and that the system consisting of its projections is equivalent to the original system of contracts.

In realistic scenarios, the existence of a contractual agreement among participants does not guarantee that progress and safety will also hold at runtime: in fact, a participant may advertise a contract promising some behaviour, and then fail to respect it – either maliciously or accidentally. Such failures may then cascade on other participants, e.g., if they remain stuck waiting for a promised message that is never sent.

This sort of situations can be modelled in the CO_2 calculus, formally defined in Section 5.4. A CO_2 system (or *network*) for the store-and-two-buyer example may be imple-

mented as follows:

$$T_{\mathtt{sb_1b_2}} \;=\; (x,y,z)\,\big(\mathtt{s}[\![\mathsf{tell}_{\mathtt{s}} \downarrow_x P_{\mathtt{s}}\,.\,\mathsf{fuse}\,.\,R_{\mathtt{s}}]\!] \;\mid\; \mathtt{b_1}[\![\mathsf{tell}_{\mathtt{s}} \downarrow_y P_{\mathtt{b_1}}\,.\,R_{\mathtt{b_1}}]\!] \;\mid\; \mathtt{b_2}[\![\mathsf{tell}_{\mathtt{s}} \downarrow_z P_{\mathtt{b_2}}\,.\,R_{\mathtt{b_2}}]\!]\big)$$

Here, participant $\mathtt{s}$ advertises its contract $P_{\mathtt{s}}$ to itself via the primitive $\mathsf{tell}_{\mathtt{s}} \downarrow_x P_{\mathtt{s}}$, where x is used as a session handle for interacting with other participants. Participants $\mathtt{b_1}$ and $\mathtt{b_2}$ advertise their respective contracts to $\mathtt{s}$ with a similar invocation.

In this example, $\mathtt{s}$ also plays the role of *contract broker*: once all the contracts have been advertised, the fuse primitive can establish a new session, based on the fact that the global agreement $\mathcal{G}_{\mathtt{sb_1b_2}}$ can be synthesised from $P_{\mathtt{s}}$, $P_{\mathtt{b_1}}$ and $P_{\mathtt{b_2}}$. This new session is shared among participants $\mathtt{s}$, $\mathtt{b_1}$ and $\mathtt{b_2}$.

At this point, the execution of the network (i.e., the continuation of processes $R_{\mathtt{s}}$, $R_{\mathtt{b_1}}$, and $R_{\mathtt{b_2}}$) is not required to respect the contracts. In fact, we will see that when the contracts are violated, the calculus allows for *culpable* participants to be always identified. Furthermore, we will discuss *honesty*, i.e., the guarantee that a participant will always fulfil its advertised contracts – even in contexts where other participants fail to fulfil theirs. When such a guarantee holds, the contractual progress and safety are also reflected in the runtime behaviour of the network.

Other compliant contracts. Our contract model allows for other scenarios. For instance, a participant $\mathtt{b_{12}}$ may impersonate both buyers, and promise to always accept the store offer, by advertising the following contract:

$$P_{\mathtt{b_{12}}} \;=\; \mathbf{s''}!\mathsf{req}\,;\mathbf{s''}!\mathsf{req}\,;\mathbf{s''}?\mathsf{quote}\,;\mathbf{s''}!\mathsf{order}\,;\mathbf{s''}?\mathsf{ok}$$

where the request to the store ($\mathbf{s''}$) is sent twice (i.e., once for each impersonated buyer). In this case, if we combine $P_{\mathtt{s}}$ and $P_{\mathtt{b_{12}}}$ with substitutions $\{\mathtt{s}/\mathbf{s''}\}$ and $\{\mathtt{b_{12}}/\mathbf{b_1},\mathbf{b_2}\}$, we can

find an agreement by synthesising the following global type:

$$\begin{aligned} \mathcal{G}_{\mathtt{sb}_{12}} \quad = \quad & \mathtt{b}_{12} \to \mathtt{s} : \mathsf{req};\ \mathtt{b}_{12} \to \mathtt{s} : \mathsf{req}; \\ & \mathtt{s} \to \mathtt{b}_{12} : \mathsf{quote};\ \mathtt{b}_{12} \to \mathtt{s} : \mathsf{order}; \\ & \mathtt{s} \to \mathtt{b}_{12} : \mathsf{ok} \end{aligned}$$

This scenario may be modelled with the following network:

$$T_{\mathtt{sb}_{12}} \quad = \quad (x,w)\big(\mathtt{s}[\![\mathsf{tell}_{\mathtt{s}} \downarrow_x P_{\mathtt{s}} \,.\, \mathsf{fuse} \,.\, R_{\mathtt{s}}]\!] \quad | \quad \mathtt{b}_{12}[\![\mathsf{tell}_{\mathtt{s}} \downarrow_w P_{\mathtt{b}_{12}} \,.\, R_{\mathtt{b}_{12}}]\!]\big)$$

where the fuse prefix can now create a session involving $\mathtt{s}$ and $\mathtt{b}_{12}$.

The participants in the networks $T_{\mathtt{sb}_1\mathtt{b}_2}$ and $T_{\mathtt{sb}_{12}}$ may also be combined, so to obtain:

$$\begin{aligned} T_{\mathtt{sb}_1\mathtt{b}_2\mathtt{b}_{12}} \quad = \quad & (x,y,z,w)\big(\ \mathtt{s}[\![\mathsf{tell}_{\mathtt{s}} \downarrow_x P_{\mathtt{s}} \,.\, \mathsf{fuse} \,.\, R_{\mathtt{s}}]\!] \\ & \qquad | \quad \mathtt{b}_1[\![\mathsf{tell}_{\mathtt{s}} \downarrow_y P_{\mathtt{b}_1} \,.\, R_{\mathtt{b}_1}]\!] \quad | \quad \mathtt{b}_2[\![\mathsf{tell}_{\mathtt{s}} \downarrow_z P_{\mathtt{b}_2} \,.\, R_{\mathtt{b}_2}]\!] \\ & \qquad | \quad \mathtt{b}_{12}[\![\mathsf{tell}_{\mathtt{s}} \downarrow_w P_{\mathtt{b}_{12}} \,.\, R_{\mathtt{b}_{12}}]\!]\ \big) \end{aligned}$$

In this case, after all contracts have been advertised to $\mathtt{s}$, either a session corresponding to $\mathcal{G}_{\mathtt{sb}_1\mathtt{b}_2}$, or to $\mathcal{G}_{\mathtt{sb}_{12}}$, may take place, thus involving a different number of participants depending on which contracts are fused.

5.3 A Choreography-Based Contract Model

We introduce a contract model based on the results of Chapter 3. Individual contracts are expressed using the syntax of local types; while contractual compliance is based on the synthesis of global types: a set of contracts is compliant if it is possible to synthesise a choreography from it.

Local types as contracts. Let $\mathcal{P}$ be a set of *participant variables* ranged over by $\mathbf{s}$, $\mathbf{r}$, $\mathbf{n}$, etc. and let $\underline{\mathtt{s}}$, $\underline{\mathtt{r}}$, $\underline{\mathtt{n}}$, etc. range over $\mathbb{P} \cup \mathcal{P}$ (recall that $\mathbb{P}$ is the set of *participant names*). The syntax of systems and contracts below is the same as in Section 3.2 but for the introduction of participant variables.

$$S, S' \quad ::= \quad S \mid S' \quad | \quad \mathtt{n}[P] \quad | \quad \mathtt{sr} : \rho \quad | \quad \mathbf{0}$$

$$P, Q \quad ::= \quad \bigoplus_{i \in I} \underline{\mathtt{r}}_i ! \mathsf{a}_i ; P_i \quad | \quad \sum_{i \in I} \underline{\mathtt{s}} ? \mathsf{a}_i ; P_i \quad | \quad \mu \chi . P \quad | \quad \chi$$

As before, we assume that there is at most one queue per pair of participants, (i.e., one channel per direction), that participant names are pairwise distinct in a system S, and that local types are closed.

We write $S \xrightarrow{\mathtt{s} \leftrightarrows \mathtt{r} : \mathsf{a}} S'$ when either $S \xrightarrow{\mathtt{s} \to \mathtt{r} : \mathsf{a}} S'$ or $S \xrightarrow{\mathtt{s} \leftarrow \mathtt{r} : \mathsf{a}} S'$, where $\mathtt{s} \to \mathtt{r} : \mathsf{a}$ indicates that $\mathtt{s}$ puts a datum on a queue $\mathtt{sr}$, and label $\mathtt{s} \leftarrow \mathtt{r} : \mathsf{a}$ indicates that $\mathtt{s}$ retrieves a datum from the queue $\mathtt{rs}$ (cf. Section 3.2).

Choreography synthesis as compliance. Essentially, in the rest of this chapter, we say that a set of contracts is *compliant*, if it can be assigned a global type, possibly after applying a set of substitutions on the contracts.

Below, we give an example to illustrate the use of local types as contracts and choreography synthesis as a compliance relation between contracts.

Example 5.1. Building up on the example from the previous section, we combine the contract of store $\mathtt{s}$ with those of customers $\mathtt{b}_1$ and $\mathtt{b}_2$, and we obtain the system:

$$\begin{aligned} S_{\mathtt{s}\mathtt{b}_1\mathtt{b}_2} &= \mathtt{s}[P_{\mathtt{s}} \{^{\mathtt{b}_1}/_{\mathbf{b_1}}\} \{^{\mathtt{b}_2}/_{\mathbf{b_2}}\}] \mid \mathtt{b}_1[P_{\mathtt{b}_1} \{^{\mathtt{s}}/_{\mathbf{s}}\} \{^{\mathtt{b}_2}/_{\mathbf{b}'_2}\}] \mid \mathtt{b}_2[P_{\mathtt{b}_2} \{^{\mathtt{s}}/_{\mathbf{s}'}\} \{^{\mathtt{b}_1}/_{\mathbf{b}'_1}\}] \\ &= \mathtt{s}[\mathtt{b}_1?\mathsf{req} ; \mathtt{b}_2?\mathsf{req} ; \mathtt{b}_1!\mathsf{quote} ; (\mathtt{b}_1?\mathsf{order} ; \mathtt{b}_2!\mathsf{ok} + \mathtt{b}_1?\mathsf{bye} ; \mathtt{b}_2!\mathsf{bye})] \\ &\quad \mid \mathtt{b}_1[\mathtt{s}!\mathsf{req} ; \mathtt{s}?\mathsf{quote} ; (\mathtt{b}_2!\mathsf{ok} ; \mathtt{s}!\mathsf{order} \oplus \mathtt{b}_2!\mathsf{bye} ; \mathtt{s}!\mathsf{bye})] \\ &\quad \mid \mathtt{b}_2[\mathtt{s}!\mathsf{req} ; (\mathtt{b}_1?\mathsf{ok} ; \mathtt{s}?\mathsf{ok} + \mathtt{b}_1?\mathsf{bye} ; \mathtt{s}?\mathsf{bye})] \end{aligned}$$

which is assigned the following global type:

$$\mathcal{G}_{\mathtt{sb_1b_2}} \;=\; \mathtt{b_1} \to \mathtt{s} : \mathsf{req};\ \mathtt{b_2} \to \mathtt{s} : \mathsf{req};\ \mathtt{s} \to \mathtt{b_1} : \mathsf{quote};\ \left(\begin{array}{c} \mathtt{b_1} \to \mathtt{b_2} : \mathsf{ok};\ \mathtt{b_1} \to \mathtt{s} : \mathsf{order};\ \mathtt{s} \to \mathtt{b_2} : \mathsf{ok} \\ + \\ \mathtt{b_1} \to \mathtt{b_2} : \mathsf{bye};\ \mathtt{b_1} \to \mathtt{s} : \mathsf{bye};\ \mathtt{s} \to \mathtt{b_2} : \mathsf{bye} \end{array} \right)$$

that is to say that $\circ \vdash S_{\mathtt{sb_1b_2}} \triangleright \mathcal{G}_{\mathtt{sb_1b_2}}$ holds.

If we combine the store $\mathtt{s}$ with $\mathtt{b}_{12}$ we have

$$\begin{array}{rcl} S_{\mathtt{sb}_{12}} & = & \mathtt{s}[P_{\mathtt{s}}\,\{^{\mathtt{b}_{12}}/_{\mathbf{b_1},\mathbf{b_2}}\}] \;\mid \mathtt{b}_{12}[P_{\mathtt{b}_{12}}\,\{^{\mathtt{s}}/_{\mathbf{s}''}\}] \\ & = & \mathtt{s}[\mathtt{b}_{12}?\mathsf{req}\,;\mathtt{b}_{12}?\mathsf{req}\,;\mathtt{b}_{12}!\mathsf{quote}\,;(\mathtt{b}_{12}?\mathsf{order}\,;\mathtt{b}_{12}!\mathsf{ok} \;+\; \mathtt{b}_{12}?\mathsf{bye}\,;\mathtt{b}_{12}!\mathsf{bye})] \\ & & \mid \mathtt{b}_{12}[\mathtt{s}!\mathsf{req}\,;\mathtt{s}!\mathsf{req}\,;\mathtt{s}?\mathsf{quote}\,;\mathtt{s}!\mathsf{order}\,;\mathtt{s}?\mathsf{ok}] \\ \mathcal{G}_{\mathtt{sb}_{12}} & = & \mathtt{b}_{12} \to \mathtt{s} : \mathsf{req};\ \mathtt{b}_{12} \to \mathtt{s} : \mathsf{req}; \\ & & \mathtt{s} \to \mathtt{b}_{12} : \mathsf{quote};\ \mathtt{b}_{12} \to \mathtt{s} : \mathsf{order}; \\ & & \mathtt{s} \to \mathtt{b}_{12} : \mathsf{ok} \end{array}$$

and, again, the judgement $\circ \vdash S_{\mathtt{sb}_{12}} \triangleright \mathcal{G}_{\mathtt{sb}_{12}}$ holds. ◇

5.4 Contract-Oriented Computing and Choreographies

We first introduce a version of the CO_2 calculus [7] adapted to multiparty contracts and sessions. Secondly, we discuss several options for fine-grained control of session establishment, which are made possible thanks to the fact that we base contract agreements on the existence of a choreography.

5.4.1 A Choreography-Based CO_2

Let $\mathbb{K}$ and $\mathcal{K}$ be disjoint sets of, respectively, *session names* (ranged over by $k, k', \ldots$) and *session variables* (ranged over by $x, y, z \ldots$). Let $u, v, \ldots$ range over $\mathbb{K} \cup \mathcal{K}$.

The syntax of CO_2 is as follows:

$$
\begin{array}{lcll}
R & ::= & \sum_{i \in I} p_i \,.\, R_i \;\mid\; R \mid R \;\mid\; (\vec{u}, \vec{\mathbf{n}})\, R \;\mid\; X(\vec{u}, \vec{\underline{\mathtt{n}}}) \;\mid\; \mathbf{0} & \text{[PROCESSES]} \\
p & ::= & \tau \;\mid\; \mathsf{tell}_{\underline{\mathtt{n}}} \downarrow_u P \;\mid\; \mathsf{fuse} \;\mid\; \mathsf{do}^{u}_{\underline{\mathtt{n}}}\, \mathsf{a} & \text{[PREFIXES]} \\
K & ::= & \downarrow_u \mathtt{n}\, \mathsf{says}\, P \;\mid\; K \mid K & \text{[LATENT CONTRACTS]} \\
T & ::= & \mathtt{n}[\![R]\!] \;\mid\; \mathtt{n}[\![K]\!] \;\mid\; k[\![S]\!] \;\mid\; T \mid T \;\mid\; (\vec{u}, \vec{\mathbf{n}})T \;\mid\; \mathbf{0} & \text{[NETWORKS]}
\end{array}
$$

CO_2 features CCS-style processes, equipped with branching $\sum$ (not to be confused with the choice operator used in contracts), parallel composition $|$, restrictions of session and participant variables, and named process invocation. We often write $R + R'$ for the binary version of $\sum$, and we consider $+$ to be an associative and commutative operator. The prefixes are for internal action (τ), contract advertisement ($\mathsf{tell}\downarrow$), session creation upon contractual agreement (fuse), and execution of contractual actions (do). A *latent contract* of the form $\downarrow_u \mathtt{n}\, \mathsf{says}\, P$ represents the promise of participant $\mathtt{n}$ to fulfil P by executing do-actions on a session variable u. A network T may be the parallel composition of processes of the form $\mathtt{n}[\![R]\!]$ (where $\mathtt{n}$ is the participant executing R), *latent* contracts $\mathtt{n}[\![K]\!]$ (where $\mathtt{n}$ is the participant to whom the contracts in K have been advertised), and established sessions $k[\![S]\!]$ (where k is a session name, and S is a system of *stipulated* contracts as in Section 5.3). Note that CO_2 process and network productions allow to delimit both session names/variables ($\vec{u}$) and participant variables ($\vec{\mathbf{n}}$), but *not* participant names, which are considered public. Hereafter, we assume that bound participant variables are pairwise distinct.

Remark 5.1. *Syntactically, the only difference between our version of CO_2 and the calculus in [4] is that, in our adaptation,* $\mathsf{do}^{u}_{\underline{\mathtt{n}}}\,\mathsf{a}$*-prefixes mention the participant towards which*

$$[\text{TELL}]\ \mathtt{s}[\![\mathsf{tell}_{\mathtt{r}}\downarrow_x P.R+R' \mid R'']\!] \ \rightarrow\ \mathtt{s}[\![R \mid R'']\!] \ \mid\ \mathtt{r}[\![\downarrow_x \mathtt{s}\,\mathsf{says}\,P]\!]$$

$$[\text{FUSE}]\ \frac{K \rhd^{\sigma}_{\pi} S \qquad \vec{\mathtt{s}} = \mathtt{dom}(\pi) \qquad \vec{u} = \mathtt{dom}(\sigma) \qquad \mathtt{img}(\sigma) = \{k\} \qquad k \text{ fresh}}{(\vec{u},\vec{\mathtt{s}})\left(\mathtt{s}[\![\mathsf{fuse}.R+R' \mid R'']\!] \mid \mathtt{s}[\![K]\!] \mid T\right) \ \rightarrow\ (k)\left(\mathtt{s}[\![R \mid R'']\!]\sigma\pi \mid k[\![S \mid \mathrm{Q}(S)]\!] \mid T\sigma\pi\right)}$$

$$[\text{DO}]\ \frac{S \xrightarrow{\mathtt{s}\leftrightharpoons\mathtt{r}:\mathtt{a}} S'}{k[\![S]\!] \ \mid\ \mathtt{s}\left[\!\left[\mathsf{do}^{k}_{\mathtt{r}}\,\mathtt{a}.R+R' \mid R''\right]\!\right] \ \rightarrow\ k[\![S']\!] \ \mid\ \mathtt{s}[\![R \mid R'']\!]}$$

$$[\text{TAU}]\ \mathtt{s}[\![\tau.R+R' \mid R'']\!] \ \rightarrow\ \mathtt{s}[\![R \mid R'']\!]$$

$$[\text{DEF}]\ \frac{X(\vec{u},\vec{\mathtt{n}}) \stackrel{def}{=} R \qquad (\vec{z},\vec{\mathtt{n}'})\left(\mathtt{s}[\![R\{\vec{v}/\vec{u},\ \vec{\mathtt{r}}/\vec{\mathtt{n}}\} \mid R']\!] \mid T\right) \ \rightarrow\ T'}{(\vec{z},\vec{\mathtt{n}'})\left(\mathtt{s}[\![X(\vec{v},\vec{\mathtt{r}}) \mid R']\!] \mid T\right) \ \rightarrow\ T'}$$

$$[\text{PAR}]\ \frac{T \rightarrow T'}{T \mid T'' \rightarrow T' \mid T''} \qquad [\text{RES}]\ \frac{T \rightarrow T'}{(\vec{u},\vec{\mathtt{s}})T \rightarrow (\vec{u},\vec{\mathtt{s}})T'}$$

Figure 5.1: Semantics rules for CO_2

the action a *is directed.*

The semantics of CO_2 is given by the rules in Figure 5.1. Rule [TELL] allows a participant s to advertise a contract P to r; as a result, a new *latent* contract is created, recording the fact that it was promised by s. Rule [FUSE] establishes a new session: the latent contracts held in $\mathtt{s}[\![K]\!]$ are combined, and their participant variables substituted, in order to find an *agreement*, i.e., a system of local types S that satisfies the relation $K \rhd^{\sigma}_{\pi} S$ (see Definition 5.1 below). Recall that $\mathrm{Q}(S)$ stands for the system of empty queues connecting all pairs of participants in S. Provided that an agreement is found, fresh session name k and participants names are shared among the parties, via substitutions σ and π; within the session, the involved contracts become *stipulated* (as opposed to "latent", before the agreement). Rule [DO] allows s to perform an input/output action a towards r on session k, provided that S permits it. Rule [TAU] allows a participant to make an internal move. Rule [DEF] deals with named process invocation. Observe that a process invocation may

contain session variables/names and participant varibales/names. Rules [PAR] and [RES] are standard.

When needed, we label network transitions: $T \xrightarrow{\mathtt{s}:\, p} T'$ means that T reduces to T' through a prefix p fired by participant $\mathtt{s}$.

Remark 5.2. *In the following, we assume that sessions, i.e., sub-networks of the form $k[\![S]\!]$, are only created at the runtime by rule* [FUSE].

Example 5.2. Consider the network:

$$T_{\text{ex5.2}} \;=\; \mathtt{s}\left[\!\!\left[\mathsf{do}^{k}_{\mathtt{r}}\,\mathsf{int} + \mathsf{do}^{k}_{\mathtt{r}}\,\mathsf{bool} \right]\!\!\right] \;\mid\; k[\![\mathtt{s}[\mathtt{r}!\mathsf{int}] \mid \mathtt{r}[\mathtt{s}?\mathsf{int}] \mid \mathtt{sr} : \varepsilon \mid \mathtt{rs} : \varepsilon]\!] \;\mid\; \mathtt{r}\left[\!\!\left[\mathsf{do}^{k}_{\mathtt{s}}\,\mathsf{int} \right]\!\!\right]$$

Here, the CO_2 process of participant $\mathtt{s}$ can perform an action towards $\mathtt{r}$ on session k, with either a message of sort int or bool. However, $\mathtt{s}$'s contract in k only specifies that $\mathtt{s}$ should send a message of sort int to $\mathtt{r}$: therefore, according to rule [DO], only the first branch of $\mathtt{s}$ may be chosen, and the network reduces as follows:

$$\begin{array}{lcl}
T_{\text{ex5.2}} & \xrightarrow{\mathtt{s}:\, \mathsf{do}^{k}_{\mathtt{r}}\,\mathsf{int}} & \mathtt{s}[\![\mathbf{0}]\!] \;\mid\; k[\![\mathtt{s}[\mathbf{0}] \mid \mathtt{r}[\mathtt{s}?\mathsf{int}] \mid \mathtt{sr} : \mathsf{int} \mid \mathtt{rs} : \varepsilon]\!] \;\mid\; \mathtt{r}\left[\!\!\left[\mathsf{do}^{k}_{\mathtt{s}}\,\mathsf{int} \right]\!\!\right] \\
 & \xrightarrow{\mathtt{r}:\, \mathsf{do}^{k}_{\mathtt{s}}\,\mathsf{int}} & \mathtt{s}[\![\mathbf{0}]\!] \;\mid\; k[\![\mathtt{s}[\mathbf{0}] \mid \mathtt{r}[\mathbf{0}] \mid \mathtt{sr} : \varepsilon \mid \mathtt{rs} : \varepsilon]\!] \;\mid\; \mathtt{r}[\![\mathbf{0}]\!]
\end{array}$$

◇

A main difference between our adaptation of CO_2 and the original presentation comes from the way we specify session establishment. We adopt the session agreement relation defined below.

Definition 5.1 (Agreement relation $K \rhd^{\sigma}_{\pi} S$)**.** *Let*

$$K \equiv \Big|_{i\in I} \downarrow_{x_i} \mathtt{s}_i\,\mathsf{says}\,P_i \qquad \textit{such that } \forall\, i \neq j \in I : \mathtt{s}_{\mathtt{i}} \neq \mathtt{s}_{\mathtt{j}}$$

and let $\pi\colon \bigcup_{i\in I} \mathtt{fv}(P_i) \to \mathbb{P}$ *and* $\sigma\colon \bigcup_{i\in I} \{x_i\} \to \mathbb{K}$ *be two substitutions mapping participant variables to participant names, and session variables to session names, respectively.*

Also, let S be the system of local types $\big|_{i\in I}\, \mathtt{s_i}[P_i]\,\pi$.

We define:

$$K \rhd^{\sigma}_{\pi} S \iff \forall i \in I\colon \forall \mathbf{n} \in \mathtt{fv}(P_i)\colon \pi(\mathbf{n}) \neq \mathtt{s_i} \quad \wedge \quad \exists\, \mathcal{G} \;:\; \circ \vdash S \blacktriangleright \mathcal{G}$$

Intuitively, a system of stipulated contracts S is constructed from a set of latent contracts K, using a substitution π that maps all the participant variables in K to the participant names in K. The definition requires that, within a contract P_i, belonging to $\mathtt{s_i}$, no participant variable is substituted by $\mathtt{s_i}$ itself. If it is possible to synthesise a global type $\mathcal{G}$ out of S, then the relation holds, and a contractual agreement exists. Note that due to the condition imposed on K, each participant may have at most one contract in a given session. Example 5.3 below illustrates Definition 5.1.

Example 5.3. Consider the following network, with $\mathtt{s}$, $\mathtt{b}_1$, $\mathtt{b}_2$ from Section 5.2, and $S_{\mathtt{sb}_1\mathtt{b}_2}$ from Example 5.1:

$$T_{\mathtt{sb}_1\mathtt{b}_2} = (x,y,z)\big(\mathtt{s}[\![\mathsf{tell}_{\mathtt{s}} \downarrow_x P_{\mathtt{s}}\,.\,\mathsf{fuse}\,.\,R_{\mathtt{s}}]\!] \mid \mathtt{b}_1[\![\mathsf{tell}_{\mathtt{s}} \downarrow_y P_{\mathtt{b}_1}\,.\,R_{\mathtt{b}_1}]\!] \mid \mathtt{b}_2[\![\mathsf{tell}_{\mathtt{s}} \downarrow_z P_{\mathtt{b}_2}\,.\,R_{\mathtt{b}_2}]\!]\big)$$

$\longrightarrow\longrightarrow\longrightarrow$

$$(x,y,z)\big(\mathtt{s}[\![\mathsf{fuse}\,.\,R_{\mathtt{s}}]\!] \mid \mathtt{s}[\![\downarrow_x \mathtt{s}\,\mathsf{says}\,P_{\mathtt{s}} \mid \downarrow_y \mathtt{b}_1\,\mathsf{says}\,P_{\mathtt{b}_1} \mid \downarrow_z \mathtt{b}_2\,\mathsf{says}\,P_{\mathtt{b}_2}]\!] \mid \mathtt{b}_1[\![R_{\mathtt{b}_1}]\!] \mid \mathtt{b}_2[\![R_{\mathtt{b}_2}]\!]\big)$$

$\xrightarrow{\mathtt{s}\colon \mathsf{fuse}}$

$$(k)T'_{\mathtt{sb}_1\mathtt{b}_2} = (k)\big(\mathtt{s}[\![R_{\mathtt{s}}]\!]\sigma\pi \;\mid\; k[\![S_{\mathtt{sb}_1\mathtt{b}_2} \mid \mathrm{Q}(S_{\mathtt{sb}_1\mathtt{b}_2})]\!] \;\mid\; \mathtt{b}_1[\![R_{\mathtt{b}_1}]\!]\sigma\pi \;\mid\; \mathtt{b}_2[\![R_{\mathtt{b}_2}]\!]\sigma\pi\big)$$

$$\text{where} \qquad \sigma = \{^{k}/_{x,y,z}\} \quad \text{and} \quad \pi = \{^{\mathtt{s}}/_{\mathbf{s},\mathbf{s}'},\, ^{\mathtt{b}_1}/_{\mathbf{b_1},\mathbf{b}'_1},\, ^{\mathtt{b}_2}/_{\mathbf{b_2},\mathbf{b}'_2}\} \tag{5.1}$$

The initial network $T_{\mathtt{sb}_1\mathtt{b}_2}$ is the one considered in Section 5.2, where all the participants are ready to advertise their respective contracts to the store $\mathtt{s}$, by using a $\mathsf{tell}_{\mathtt{s}}\downarrow$-primitive. This has the effect of creating corresponding latent contracts within $\mathtt{s}$. Once all the latent contracts are in a same location, they may be fused.

$$(\vec{u},\vec{\mathbf{r}})\mathtt{s}[\![(\vec{v},\vec{\mathbf{n}})P]\!] \equiv (\vec{u},\vec{\mathbf{r}})(\vec{v},\vec{\mathbf{n}})\mathtt{s}[\![P]\!] \qquad \mathtt{s}[\![\mathbf{0}]\!] \equiv \mathbf{0} \qquad \mathtt{s}[\![K]\!] \mid \mathtt{s}[\![K']\!] \equiv \mathtt{s}[\![K \mid K']\!]$$

$$Z \mid \mathbf{0} \equiv Z \qquad Z \mid Z' \equiv Z' \mid Z \qquad (Z \mid Z') \mid Z'' \equiv Z \mid (Z' \mid Z'')$$

$$Z \mid (\vec{u},\vec{\mathbf{n}})Z' \equiv (\vec{u},\vec{\mathbf{n}})(Z \mid Z') \text{ if } \vec{u} \cap \mathtt{fnv}(Z) = \vec{\mathbf{n}} \cap \mathtt{fnv}(Z) = \varnothing$$

$$(\vec{u},\vec{\mathbf{n}})(\vec{v},\vec{\mathbf{n}})Z \equiv (\vec{v},\vec{\mathbf{n}})(\vec{u},\vec{\mathbf{n}})Z \qquad (\vec{u},\vec{\mathbf{n}})(\vec{v},\vec{\mathbf{n}})Z \equiv (\vec{u}\,\vec{v},\vec{\mathbf{n}}\,\vec{\mathbf{n}})Z$$

$$(\vec{u},\vec{\mathbf{n}})Z \equiv Z \text{ if } \vec{u} \cap \mathtt{fnv}(Z) = \vec{\mathbf{n}} \cap \mathtt{fnv}(Z) = \varnothing$$

Figure 5.2: Congruence rules for CO_2

In this case, given σ and π as in (5.1), Definition 5.1 is indeed applicable; we already saw that the system

$$\mathtt{s}[P_{\mathtt{s}}\pi] \mid \mathtt{b}_1[P_{\mathtt{b}_1}\pi] \mid \mathtt{b}_2[P_{\mathtt{b}_2}\pi] \tag{5.2}$$

may be assigned a global type. Hence, a new session k is created, based on the system of contracts in (5.2) and the queues connecting all pairs of participants.

The session variables of the latent contracts being fused (i.e., x for participant $\mathtt{s}$, y for $\mathtt{b}_1$, and z for $\mathtt{b}_2$) are all substituted with the fresh session name k in the processes $R_{\mathtt{s}}$, $R_{\mathtt{b}_1}$ and $R_{\mathtt{b}_2}$, via σ. Similarly for participant variables which are substituted with participant names, via π. ◇

The semantics of CO_2 is to be considered up-to a standard structural congruence relation $\equiv$, given in Figure 5.2, where Z, Z', Z'' range over processes, networks, and latent contracts. We write $\mathtt{fnv}(Z)$ for the free session and participant variables in Z. Remarkably, the rule

$$\mathtt{s}[\![K]\!] \mid \mathtt{s}[\![K']\!] \equiv \mathtt{s}[\![K \mid K']\!]$$

allows to select subsets of latent contracts before invoking a fuse primitive. This allows to (momentarily) disregard some contracts when searching for a global type to be synthesised. We illustrate this in Example 5.4 below.

Example 5.4. Consider the network:

$$\ldots \mathtt{r}[\![\mathsf{fuse}.R \mid R']\!] \mid \mathtt{r}[\![\downarrow_x \mathtt{s}_1 \,\mathsf{says}\, \mathbf{n_1}!\mathsf{int} \mid \downarrow_y \mathtt{s}_2 \,\mathsf{says}\, \mathbf{n_2}?\mathsf{int} \mid \downarrow_z \mathtt{s}_3 \,\mathsf{says}\, \mathbf{n_3}?\mathsf{bool}]\!] \mid \ldots$$

The fuse prefix cannot be fired: no contract matches $\mathtt{s}_3$'s, and thus, together, the three latent contracts cannot be assigned a global type. However, by rearranging the network with congruence $\equiv$, we have:

$$\ldots \mathtt{r}[\![\mathsf{fuse}.R \mid R']\!] \mid \mathtt{r}[\![\downarrow_x \mathtt{s}_1 \,\mathsf{says}\, \mathbf{n_1}!\mathsf{int} \mid \downarrow_y \mathtt{s}_2 \,\mathsf{says}\, \mathbf{n_2}?\mathsf{int}]\!] \mid \mathtt{r}[\![\downarrow_z \mathtt{s}_3 \,\mathsf{says}\, \mathbf{n_3}?\mathsf{bool}]\!] \mid \ldots$$

It is now possible to synthesise a global type $\mathtt{s}_1 \to \mathtt{s}_2 : \mathsf{int}$, and a session may be created for $\mathtt{s}_1$ and $\mathtt{s}_2$. The latent contract of $\mathtt{s}_3$ may be fused later on. $\diamond$

In Example 5.5 below, we return to the running example of Chapter 3 and show that introducing participant variables adds flexibility to the synthesis of choreographies.

Example 5.5. We reuse the local types from the running example of Chapter 3 (cf. Section 3.1), and consider them as contracts advertised by their respective participants, these contracts may now feature participant variables.

Recall that the system consists of two buyers ($\mathtt{b}_1$ and $\mathtt{b}_2$) and two sellers ($\mathtt{s}_1$ and $\mathtt{s}_2$). The expected overall behaviour is that the two buyers should agree on purchasing concurrently two items from two sellers, and then try to obtain the best price for these items. The contracts are as follows:

$$\begin{array}{lcll} P_{\mathtt{b}_1} & = & \mathtt{b}_2!\mathsf{agreement}\,;\mathtt{b}_2?\mathsf{ack}\,;Q_1 & \\ P_{\mathtt{b}_2} & = & \mathtt{b}_1?\mathsf{agreement}\,;\mathtt{b}_1!\mathsf{ack}\,;Q_2 & \\ Q_{\mathtt{i}} & = & \mathbf{s_i}!\mathsf{request}\,;\mathbf{s_i}?\mathsf{quote}\,;\mu\chi.(\mathbf{s_i}!\mathsf{ok} \oplus \mathbf{s_i}!\mathsf{no}\,;\mathbf{s_i}?\mathsf{quote}\,;\chi) & \mathtt{i} \in \{1,2\} \\ S_{\mathtt{s_i}} & = & \mathbf{b_i}?\mathsf{request}\,;\mathbf{b_i}!\mathsf{quote}\,;\mu\chi'.(\mathbf{b_i}?\mathsf{ok} \,+\, \mathbf{b_i}?\mathsf{no}\,;\mathbf{b_i}!\mathsf{quote}\,;\chi') & \mathtt{i} \in \{1,2\} \end{array} \tag{5.3}$$

Observe that the contracts now features four *participant variables*: $\mathbf{b_i}$ and $\mathbf{s_i}$ ($\mathtt{i} \in \{1,2\}$).

A network for these four participants may be implemented as follows:

$$T_{\text{ex5.5}} \;=\; (x,y,z,w)\big(\;\; \mathtt{b}_1[\![\mathsf{tell}_{\mathtt{b}_1}\downarrow_x P_{\mathtt{b}_1}\,.\,\mathsf{fuse}\,.\,R_{\mathtt{b}_1}]\!] \;\mid\; \mathtt{b}_2[\![\mathsf{tell}_{\mathtt{b}_1}\downarrow_y P_{\mathtt{b}_2}\,.\,R_{\mathtt{b}_2}]\!] \;\mid\; \mathtt{s}_1[\![\mathsf{tell}_{\mathtt{b}_1}\downarrow_z P_{\mathtt{s}_1}\,.\,R_{\mathtt{s}_1}]\!] \;\mid\; \mathtt{s}_2[\![\mathsf{tell}_{\mathtt{b}_1}\downarrow_w P_{\mathtt{s}_2}\,.\,R_{\mathtt{s}_2}]\!]\big)$$

where each participant advertises its contract to $\mathtt{b}_1$. Interestingly, the pairs buyer/seller are not determined in advance, i.e., we may have that $\mathtt{b}_1$ buys an item from $\mathtt{s}_2$, because two sets of substitutions are possible:

$$\pi_1 = \{{}^{\mathtt{s}_1}/_{\mathbf{s_1}},\ {}^{\mathtt{s}_2}/_{\mathbf{s_2}},\ {}^{\mathtt{b}_1}/_{\mathbf{b_1}},\ {}^{\mathtt{b}_2}/_{\mathbf{b_2}}\} \qquad \text{or} \qquad \pi_2 = \{{}^{\mathtt{s}_1}/_{\mathbf{s_2}},\ {}^{\mathtt{s}_2}/_{\mathbf{s_1}},\ {}^{\mathtt{b}_1}/_{\mathbf{b_2}},\ {}^{\mathtt{b}_2}/_{\mathbf{b_1}}\}$$

Now, assume that the contracts of $\mathtt{b}_1$ and $\mathtt{s}_1$ are slightly modified so that $\mathtt{b}_1$ wants to buy a book, while the seller $\mathtt{s}_1$ sells only books (i.e., it only accepts book requests). We have the following two new contracts:

$$\begin{aligned} P_{\mathtt{b}_1} &= \mathtt{b}_2!\mathsf{agreement}\,;\mathtt{b}_2?\mathsf{ack}\,;Q_1 \\ Q_1 &= \mathbf{s_1}!\mathsf{book}\,;\mathbf{s_1}?\mathsf{quote}\,;\mu\chi.(\mathbf{s_1}!\mathsf{ok} \oplus \mathbf{s_1}!\mathsf{no}\,;\mathbf{s_1}?\mathsf{quote}\,;\chi) \\ S_{\mathtt{s}_1} &= \mathbf{b_1}?\mathsf{book}\,;\mathbf{b_1}!\mathsf{quote}\,;\mu\chi'.(\mathbf{b_1}?\mathsf{ok} + \mathbf{b_1}?\mathsf{no}\,;\mathbf{b_1}!\mathsf{quote}\,;\chi') \end{aligned}$$

If the contracts of $\mathtt{b}_2$ and $\mathtt{s}_2$ are the same as in (5.3), the fusion of these contracts is deterministic: $\mathbf{b_i}$ will be instantiated to $\mathtt{b}_\mathtt{i}$ and $\mathbf{s_i}$ will be instantiated to $\mathtt{s}_\mathtt{i}$ (for $\mathtt{i} \in \{1,2\}$). Indeed, this is the only substitution that will allow a global type to be synthesised from the contracts. ◇

5.4.2 On the Flexibility of Session Establishment

We highlight the flexibility of our definition of contract agreement, together with the semantics of CO_2, by discussing three features: (*i*) contracts may use a mix of participant names and variables, (*ii*) different contracts may use common participant variables, and

(*iii*) the definition of agreement may be easily accommodated to different requirements.

Mixing participant variables and names

Mixing participant variables and names in contracts allows us to design fine-grained contracts and specify relationships between them.

- **Contracts with both participant names and variables.** A seller s may want to sell an item to a *specific* buyer b, via *any* shipping company that provides a package tracking system. The contract of s may be:

$$\mathtt{b}!\mathsf{price}\,;\mathtt{b}?\mathsf{ack}\,;\mathbf{n}!\mathsf{request}\,;\mathbf{n}?\mathsf{tracking}\,;\mathtt{b}!\mathsf{tracking}$$

  saying that the seller s must send a price to the buyer b; once b has acknowledged, s must send a shipping request to a shipper **n**; who must send back a tracking number, which is then forwarded to b. This contract may be fused only if b takes part in the session, while the role of shipper **n** may be played by any participant (offering a compliant contract).

- **Contracts sharing participant variables.** Consider the network:

$$\begin{aligned}&\mathtt{s}[\![\mathsf{tell}_{\mathtt{s}}{\downarrow_x}(\mathbf{n}!\mathsf{request}\,;\mathbf{n}?\mathsf{ack})\ldots X(\vec{z},\mathbf{n})]\!] \mid \ldots\\ &\qquad\text{where } X(\vec{z},\mathbf{n}) \stackrel{def}{=} (y,\mathbf{b})\mathsf{tell}_{\mathtt{s}}{\downarrow_y}(\mathbf{b}!\mathsf{quote}\,;\ldots\mathbf{n}!\mathsf{address})\ldots X(\vec{z},\mathbf{n})\end{aligned}$$

  Here, s advertises two contracts: the first one ($\mathbf{n}!\mathsf{request}\,;\mathbf{n}?\mathsf{ack}$) is used by s to find a shipping company, and the second ($\mathbf{b}!\mathsf{quote}\,;\ldots\mathbf{n}!\mathsf{address}$) to sell items. The two contracts are linked by the common variable **n**: whenever the first one is fused, variable **n** is instantiated to a participant name, say `shipper`, which is also substituted in the second. This means that whenever a new selling session starts, `shipper` will also be involved as the receiver of the address message.

- **Injective substitution.** Definition 5.1 is quite liberal with respect to the substitution mapping participant variables to participant names (π). The definition may be refined easily to, e.g., force π to be an injective substitution and therefore not allowing two participant variables to be mapped to the same participant identifier.

  In the example of Section 5.2, such a variation would allow the seller to disregard the second set of contracts (where one participant impersonates both buyers).

Parametrised session establishment

The participants firing fuse primitives are playing the role of brokers in our framework. Depending on their implementation, brokers may also have some obligations in the contracts they fuse, or they may want to enforce some general policy – therefore they may have additional requirements before agreeing to start a session. For instance, a broker may not want to start a session with too many participants as it may be too resource demanding (too many connections etc.). Another broker may want to start sessions that terminates within a limited number of interactions, because it has a short life expectancy, e.g., due to an approaching scheduled maintenance. Another kind of broker may precisely want to start sessions which do not terminate, e.g., if the broker is interested in resilient services. Several variations of the fuse primitive are possible thanks to the fact that we base contract agreements on objects representing the overall choreography.

In this section, we discuss some of these variations, starting from simple ones to more complex ones. The overall idea of these variation is to parametrise the fuse primitive with a function, say $\mathtt{F}^{\mathsf{fuse}}$, that takes as input a global type $\mathcal{G}$ and returns a boolean value.[1] Definition 5.1 stays essentially the same but for the slightly refined condition:

$$\exists\, \mathcal{G} \;:\; \circ \vdash S \triangleright \mathcal{G} \quad \wedge \quad \mathtt{F}^{\mathsf{fuse}}(\mathcal{G})$$

[1] We consider $\mathtt{F}^{\mathsf{fuse}}$ to be a computable function.

Remark 5.3. *This sort of variations does not affect the results that follow, since the original* fuse *primitive is also blocking. The variations only restrict the sets of compliant contracts.*

Some variations. We introduce $\mathsf{fuse}[n]$, a version of fuse that only fuses sessions where there are at least n participants; fuse_T, which has the additional constraint that no recursive behaviour is allowed in the synthesised choreography (therefore ensuring that the session will eventually terminate), and fuse_R, which only creates sessions when the synthesised choreography never terminates (i.e., it only consists of recursive behaviours).

Such extensions may be defined directly via simple definitions of function $\mathtt{F}^{\mathsf{fuse}}$:

- for $\mathsf{fuse}[n]$, we define $\mathtt{F}^{\mathsf{fuse}}(\mathcal{G}) \stackrel{def}{=} |\mathcal{P}(\mathcal{G})| \geqslant n$, recall that $\mathcal{P}(\mathcal{G})$ is the set of participants in $\mathcal{G}$;
- for fuse_T, we define $\mathtt{F}^{\mathsf{fuse}}(\mathcal{G}) \stackrel{def}{=} \mathtt{bv}(\mathcal{G}) = \varnothing$, i.e., the condition holds only if there is no recursion variable χ in $\mathcal{G}$ (recall that synthesised global types are closed since we assumed the contracts to be closed);
- for fuse_R, we define $\mathtt{F}^{\mathsf{fuse}}(\mathcal{G})$ so that it holds only if $\mathbf{0}$ is not a sub-term of $\mathcal{G}$, i.e., we add the condition that $\mathbf{0}$ does not appear in $\mathcal{G}$.

Observe that this kind of properties must be checked for at the global level because it cannot always be decided by looking at the individual contracts. For instance, a participant might exhibit a recursive behaviour in one of the branches of an external choice, while the participant it interacts with may always choose a branch that is not recursive. Consider for instance, the following system of contracts:

$$\mathtt{s}[\mathtt{r}!\mathtt{a}] \;\mid\; \mathtt{r}[\mu\chi.(\mathtt{s}?\mathtt{a} + \mathtt{s}?\mathtt{b};\chi)]$$

Looking only at the contract of $\mathtt{r}$, it appears to be recursive. However, when composed

with the contract of $\mathtt{s}$, their global type is $\mathtt{s} \to \mathtt{r} : \mathtt{a}$, i.e., a non-recursive choreography.

Sharing knowledge. In order to illustrate the expressive power of our agreement relation – thanks to the fact that we base it on an object representing the choreography of the session – we give a more advanced version of $\mathtt{F}^{\mathsf{fuse}}$ that checks semantic properties of the synthesised global type.

Consider a broker whose sole purpose is to fuse contracts only if their composition corresponds to a choreography such that all the participants share the same knowledge once the session has terminated (i.e, when all the participants have fulfilled their contracts in the session). Assume that $\mathtt{k}$ is a sort representing the knowledge of a participant; and that a sending operation $\mathtt{r}!\mathtt{k}$ made by a participant $\mathtt{s}$ means that $\mathtt{s}$ sends *all* its *current* knowledge to $\mathtt{r}$, and that a receiving operation $\mathtt{s}?\mathtt{k}$ made by a participant $\mathtt{r}$ means that $\mathtt{r}$ acquires all the knowledge that $\mathtt{s}$ has (when $\mathtt{s}$ sent the message). We need to check that, given a set of contracts, all participants share the same knowledge at the end of the session. For instance, the contracts $P_{\mathtt{s}}$ and $P_{\mathtt{r}}$ below comply with the condition, but $\overline{P_{\mathtt{s}}}$ and $\overline{P_{\mathtt{r}}}$ do not.

$$P_{\mathtt{s}} = \mathtt{r}!\mathtt{k};\mathtt{r}?\mathtt{k} \text{ and } P_{\mathtt{r}} = \mathtt{s}?\mathtt{k};\mathtt{s}!\mathtt{k} \quad \checkmark \qquad \overline{P_{\mathtt{s}}} = \mathtt{r}!\mathtt{k} \text{ and } \overline{P_{\mathtt{r}}} = \mathtt{s}?\mathtt{k} \quad \text{✘}$$

Indeed, on the right-hand side, $\mathtt{s}$ does not acquire $\mathtt{r}$'s knowledge. Note that both pairs of contract may be assigned a global type.

We introduce a map $\mathtt{K} : \mathbb{P} \to 2^{\mathbb{P}}$ that, given a participant name $\mathtt{n}$, returns the set of names representing the participants from whom $\mathtt{n}$ acquired knowledge. Initially, we have that $\mathtt{K}(\mathtt{n}) = \{\mathtt{n}\}$ for each participant $\mathtt{n}$, i.e., a participant has only access to its own knowledge. Given a global type $\mathcal{G}$, we would like that

$$\forall\, \mathtt{n} \in \mathcal{P}(\mathcal{G}) : \mathtt{K}(\mathtt{n}) = \mathcal{P}(\mathcal{G})$$

holds at the end of the session embodied by $\mathcal{G}$.

We define the boolean function $\hat{\mathtt{K}}(\mathcal{G}, N, \mathtt{K})$ which returns true if all the participants in $N \supseteq \mathcal{P}(\mathcal{G})$ share the same knowledge (with respect to $\mathtt{K}$) at the end of the session. We write $\{\text{-}/\text{-}\}$ for the update operation on maps, and $\cdot$ for the disjoint union of maps.

Definition 5.2. *Let* $\hat{\mathtt{K}}(\mathcal{G}) \stackrel{def}{=} \hat{\mathtt{K}}(\mathcal{G}, \mathcal{P}(\mathcal{G}), \mathtt{K}_0)$*, where* $\mathtt{K}_0$ *is the initial knowledge map, i.e.* $\forall \mathtt{n} \in \mathcal{P}(\mathcal{G}) : \mathtt{K}_0(\mathtt{n}) = \{\mathtt{n}\}$*, and*

$$\hat{\mathtt{K}}(\mathcal{G}, N, \mathtt{K}) \stackrel{def}{=} \begin{cases} \hat{\mathtt{K}}(\mathcal{G}', N, \mathtt{K}'), & \text{if } \mathcal{G} = \mathtt{s} \to \mathtt{r} : \mathtt{k};\ \mathcal{G}' \text{ and } \mathtt{K}' = \mathtt{K}\{{}^{\mathtt{K}(\mathtt{r}) \cup \mathtt{K}(\mathtt{s})}/_{\mathtt{r}}\} \\ \hat{\mathtt{K}}(\mathcal{G}_1, N, \mathtt{K}) \wedge \hat{\mathtt{K}}(\mathcal{G}_2, N, \mathtt{K}), & \text{if } \mathcal{G} = \mathcal{G}_1 + \mathcal{G}_2 \\ \hat{\mathtt{K}}(\mathcal{G}_1, N, \mathtt{K}_1) \wedge \hat{\mathtt{K}}(\mathcal{G}_2, N, \mathtt{K}_2), & \text{if } \mathcal{G} = \mathcal{G}_1 \mid \mathcal{G}_2 \text{ and } \mathtt{K} = \mathtt{K}_1 \cdot \mathtt{K}_2 \\ \forall \mathtt{n} \in \mathtt{dom}(\mathtt{K}) : \mathtt{K}(\mathtt{n}) = N, & \text{if } \mathcal{G} = \mathbf{0} \\ \textbf{\textit{false}}, & \text{otherwise} \end{cases}$$

The first case of Definition 5.2 handles interactions: $\mathtt{r}$ acquires all the current knowledge of $\mathtt{s}$. The case of choice simply requires that the condition is satisfied in both branches. In the case of concurrent branches, we need to partition the knowledge map in two – since the sets of participants in the concurrent branches of $\mathcal{G}$ are disjoint. In the case of $\mathcal{G} = \mathbf{0}$, we check that all participants share the same knowledge. For the sake of simplicity, the function is undefined for recursive global type.

We may use $\hat{\mathtt{K}}(\text{-})$ in session establishment with $\mathtt{F}^{\mathsf{fuse}}(\mathcal{G}) \iff \hat{\mathtt{K}}(\mathcal{G})$. We illustrate Definition 5.2 in Example 5.6 below.

Example 5.6. Consider the following contracts

$$\begin{aligned} P_{\mathtt{s}} &= \mathtt{r}_1?\mathtt{k}; \mathtt{r}_2?\mathtt{k}; \mathtt{r}_3?\mathtt{k}; \mathtt{r}_1!\mathtt{k}; \mathtt{r}_2!\mathtt{k}; \mathtt{r}_3!\mathtt{k} \\ P_{\mathtt{r}_\mathtt{i}} &= \mathtt{s}!\mathtt{k}; \mathtt{s}?\mathtt{k} \qquad \mathtt{i} \in \{1,2,3\} \end{aligned}$$

The composition of these contracts, i.e.

$$\mathtt{s}[P_{\mathtt{s}}] \mid \mathtt{r}_1[P_{\mathtt{r}_1}] \mid \mathtt{r}_2[P_{\mathtt{r}_2}] \mid \mathtt{r}_3[P_{\mathtt{r}_3}]$$

gives us the following global type

$$\begin{aligned} \mathcal{G}_{\text{ex5.6}} \;=\; & \mathtt{r}_1 \to \mathtt{s}:\mathtt{k}; \mathtt{r}_2 \to \mathtt{s}:\mathtt{k}; \mathtt{r}_3 \to \mathtt{s}:\mathtt{k}; \\ & \mathtt{s} \to \mathtt{r}_1:\mathtt{k}; \mathtt{s} \to \mathtt{r}_2:\mathtt{k}; \mathtt{s} \to \mathtt{r}_3:\mathtt{k}; \mathbf{0} \end{aligned}$$

$\hat{\mathtt{K}}(\mathcal{G}_{\text{ex5.6}})$ holds since at the end of the first line of $\mathcal{G}_{\text{ex5.6}}$, we have $\mathtt{K}(\mathtt{s}) = \{\mathtt{s}, \mathtt{r}_1, \mathtt{r}_2, \mathtt{r}_3\}$ and, in the second line, $\mathtt{s}$ shares its knowledge with all the other participants. ◇

Interestingly, it is possible to refine session establishments so that only "efficient" contracts are fused. For instance, in Example 5.6, the sharing of knowledge is done in a very centralised way: $\mathtt{s}$ acquires the knowledge of all the other participants, then share back all $\mathtt{s}$ knows.

One may compute a bound on the number of consecutive interactions that are required for n participants to share their knowledge [48]. If the depth of the parsing tree of the synthesised global type is greater than this bound, then the contracts are not "efficient". We give a more efficient set of contracts in Example 5.7 below.

Example 5.7. Consider the following contracts

$$\begin{aligned} P_{\mathtt{s}} &= \mathtt{r}_1?\mathtt{k}; \mathtt{r}_2?\mathtt{k}; \mathtt{r}_3?\mathtt{k}; \mathtt{r}_1!\mathtt{k}; \mathtt{r}_3!\mathtt{k} & \qquad P_{\mathtt{r}_2} &= \mathtt{s}!\mathtt{k}; \mathtt{r}_1?\mathtt{k} \\ P_{\mathtt{r}_1} &= \mathtt{s}!\mathtt{k}; \mathtt{s}?\mathtt{k}; \mathtt{r}_2!\mathtt{k} & \qquad P_{\mathtt{r}_3} &= \mathtt{s}!\mathtt{k}; \mathtt{s}?\mathtt{k} \end{aligned}$$

where the difference between these contracts and those in Example 5.6, is that $\mathtt{r}_1$ is now the one who shares the overall knowledge to participant $\mathtt{r}_2$. The composition of these

contracts gives us the following global type:

$$\begin{aligned}\mathcal{G}_{\text{ex5.7}} \;=\;& \mathtt{r}_1 \to \mathtt{s}:\mathtt{k};\mathtt{r}_2 \to \mathtt{s}:\mathtt{k};\mathtt{r}_3 \to \mathtt{s}:\mathtt{k};\\ & \mathtt{s} \to \mathtt{r}_1:\mathtt{k};(\mathtt{r}_1 \to \mathtt{r}_2:\mathtt{k};\mathbf{0} \mid \mathtt{s} \to \mathtt{r}_3:\mathtt{k};\mathbf{0})\end{aligned}$$

We have again that $\mathtt{s}$ has all the knowledge at the end of the first line of $\mathcal{G}_{\text{ex5.7}}$, but in this case $\mathtt{s}$ and $\mathtt{r}_1$ share back their knowledge concurrently to $\mathtt{r}_3$ and $\mathtt{r}_2$, respectively. Thus, $\mathcal{G}_{\text{ex5.7}}$ is (slightly) more efficient than $\mathcal{G}_{\text{ex5.6}}$. ◇

5.5 The Problem of Honesty

In this section, we discuss and define the notion of *honesty* [7], i.e., the ability of a participant to always fulfil its contracts, in any context. In our contract-oriented setting, honesty is essentially the counterpart of well-typedness in a session type setting: the static proof that a participant always honours its contracts provides guarantees about its runtime behaviour.

As seen in Example 5.2, each do prefix within the process of a participant, say $\mathtt{s}[\![P]\!]$, is driven by the contract that $\mathtt{s}$ promised to abide by. In a sense, CO_2 is *culpability-driven*, according to Definition 5.3 below: when a participant is "culpable", it has the duty of making the session progress according to its contract.

Definition 5.3 (Culpability)**.** *Let T be a network with a session k, i.e.,*

$$T \equiv (\vec{u},\vec{\mathbf{n}})\,(\mathtt{s}[\![R]\!] \mid k[\![S]\!] \mid T_1) \mid T_2$$

We say that $\mathtt{s}$ *is* culpable in T *when there exist* $\mathtt{r}$ *and* $\mathtt{a}$ *such that* $S \xrightarrow{\mathtt{s}\leftrightharpoons\mathtt{r}:\mathtt{a}}$.

A culpable participant can overcome its status by firing its do prefixes, according to rule [DO], until another participant becomes culpable or session k terminates. Hence, as long as a culpable participant $\mathtt{s}$ does not enable a do-prefix matching a contractual action,

s will remain culpable. Note that when a participant is involved in multiple sessions, it may end up being culpable in more than one of them.

When a participant s is always able to fulfil its contractual actions (i.e., overcome its culpability), no matter what other participants do, then it is said to be *honest* (cf. Definition 5.8). This is a desirable property in contract-oriented scenarios: a participant may be stuck in a culpable condition either due to "simple" bugs (cf. Example 5.9), or due to the unexpected (or malicious) behaviour of other participants (cf. Example 5.12). Therefore, before deploying a service, its developers may want to ensure that it will always be able to exculpate itself.

Formally, as in [4], we base the definition of honesty on the relationship between the ready set of a contract and the ready set of a CO_2 process. We call the former *contract ready set* and the latter *process ready set*. The concept of contract ready set is similar to [4, 7, 30], where only bilateral contracts are considered. Here, we adapt it to suit our multiparty contract model, the main difference being that we take into account towards which participant an action is directed.

Definition 5.4 (Contract ready set)**.** *The* ready set of a contract P*, written* $\mathtt{Crs}(P)$*, is:*

$$\mathtt{Crs}(P) = \begin{cases} \mathtt{Crs}(P') & \text{if } P = \mu\chi.P' \\ \{\{(\mathtt{r_i}, \mathtt{a_i})\} \mid i \in I\} & \text{if } P = \bigoplus_{i\in I} \mathtt{r_i}!\mathtt{a_i};P_i \quad \text{and} \quad I \neq \varnothing \\ \{\{(\mathtt{s}, \mathtt{a_i}) \mid i \in I\}\} & \text{if } P = \sum_{i\in I} \mathtt{s}?\mathtt{a_i};P_i \end{cases}$$

Intuitively, when a participant n is bound to a contract P, the ready set of P tells which interactions n must be able to perform towards other participants. Each interaction has the form of a pair, consisting of a participant name and a message sort. The interactions offered by an external choice are all available at once, while those offered by an internal choice are mutually exclusive.[2]

[2] Recall that for local types, **0** is defined as an external/internal choice where $I = \varnothing$ (cf. Section 3.2).

Example 5.8. Consider the system of contracts $S_{\mathtt{s}\mathtt{b}_1\mathtt{b}_2}$ from Example 5.1 and, in particular, its stipulated contracts, with substitution $\pi = \{\mathtt{s}/_{\mathbf{s},\mathbf{s}'}, \mathtt{b}_1/_{\mathbf{b_1},\mathbf{b}'_1}, \mathtt{b}_2/_{\mathbf{b_2},\mathbf{b}'_2}\}$ from Example 5.3:

$$\begin{aligned}
\hat{P}_{\mathtt{s}} &= P_{\mathtt{s}}\pi = \mathtt{b}_1?\mathsf{req};\mathtt{b}_2?\mathsf{req};\mathtt{b}_1!\mathsf{quote};(\mathtt{b}_1?\mathsf{order};\mathtt{b}_2!\mathsf{ok} + \mathtt{b}_1?\mathsf{bye};\mathtt{b}_2!\mathsf{bye}) \\
\hat{P}_{\mathtt{b}_1} &= P_{\mathtt{b}_1}\pi = \mathtt{s}!\mathsf{req};\mathtt{s}?\mathsf{quote};(\mathtt{b}_2!\mathsf{ok};\mathtt{s}!\mathsf{order} \oplus \mathtt{b}_2!\mathsf{bye};\mathtt{s}!\mathsf{bye}) \\
\hat{P}_{\mathtt{b}_2} &= P_{\mathtt{b}_2}\pi = \mathtt{s}!\mathsf{req};(\mathtt{b}_1?\mathsf{ok};\mathtt{s}?\mathsf{ok} + \mathtt{b}_1?\mathsf{bye};\mathtt{s}?\mathsf{bye})
\end{aligned}$$

We have $\mathtt{Crs}\big(\hat{P}_{\mathtt{s}}\big) = \{\{(\mathtt{b}_1, \mathsf{req})\}\}$: in other words, at this point of the contract, an interaction is expected between $\mathtt{s}$ and $\mathtt{b}_1$ (since $\mathtt{s}$ is waiting for req), while no interaction is expected between $\mathtt{s}$ and $\mathtt{b}_2$.

When $\hat{P}_{\mathtt{s}}$ reaches its external choice, we have:

$$\hat{P}'_{\mathtt{s}} = \mathtt{b}_1?\mathsf{order};\mathtt{b}_2!\mathsf{ok} + \mathtt{b}_1?\mathsf{bye};\mathtt{b}_2!\mathsf{bye}$$

Now, the ready set becomes $\mathtt{Crs}\big(\hat{P}'_{\mathtt{s}}\big) = \{\{(\mathtt{b}_1, \mathsf{order}), (\mathtt{b}_1, \mathsf{bye})\}\}$, i.e., $\mathtt{s}$ must handle both answers from $\mathtt{b}_1$. Instead, when $\hat{P}_{\mathtt{b}_1}$ reduces to its internal choice, we have:

$$\hat{P}'_{\mathtt{b}_1} = \mathtt{b}_2!\mathsf{ok};\mathtt{s}!\mathsf{order} \oplus \mathtt{b}_2!\mathsf{bye};\mathtt{s}!\mathsf{bye}$$

Thus, its ready set becomes $\mathtt{Crs}\big(\hat{P}'_{\mathtt{b}_1}\big) = \{\{(\mathtt{b}_2, \mathsf{ok})\}, \{(\mathtt{b}_2, \mathsf{bye})\}\}$: $\mathtt{b}_1$ is free to choose either branch. ◇

Example 5.8 shows that when a *contract P* of a participant evolves within a system *S* its ready set changes. Now we define the counterpart of contract ready set for CO_2 processes, i.e., the *process ready set*. Again, we adapt the definition from [4] to our multiparty contract model.

Definition 5.5 (Process ready set)**.** *For all networks T, all participants* $\mathtt{s}, \mathtt{r}$ *and sessions*

u, we define the set of pairs:

$$\mathtt{Prs}^{u}_{\mathtt{s}}(T) = \left\{ (\mathtt{r},\mathtt{a}) \;\middle|\; \begin{array}{c} \exists\, \vec{v},\vec{\mathbf{n}},R,R',R'',T' : \\ T \equiv (\vec{v},\vec{\mathbf{n}})\,(\mathtt{s}[\![\mathsf{do}^{u}_{\mathtt{r}}\,\mathtt{a}\,.\,R+R' \mid R'']\!] \mid T_0) \mid T_1 \;\;\wedge\;\; u \notin \vec{v} \end{array} \right\}$$

Intuitively, Definition 5.5 says that the process ready set of $\mathtt{s}$ over a session u in a network T contains the interactions that $\mathtt{s}$ is immediately able to perform with other participants through its do-actions on u. As in a contract ready set, the interactions are represented by participant/sort pairs.

Next, we want to characterise a weaker notion of the process ready set, so it only takes into account the first actions *on a specific session* that a participant is ready to make.

Definition 5.6 (Weak Process ready set)**.** *We write* $T \xrightarrow{\neq(\mathtt{s}:\,\mathsf{do}^{u})} T'$ *iff:*

$$\exists\, \mathtt{s}', p :\; T \xrightarrow{\mathtt{s}':\,p} T' \implies \big(\mathtt{s} \neq \mathtt{s}' \quad\vee\quad \forall\, \mathtt{a} :\; \forall\, \mathtt{r} :\; p = \mathsf{do}^{v}_{\mathtt{r}}\,\mathtt{a} \implies u \neq v\big)$$

We define the set of pairs $\mathtt{WPrs}^{u}_{\mathtt{s}}(T)$ *as:*

$$\mathtt{WPrs}^{u}_{\mathtt{s}}(T) \;=\; \left\{ (\mathtt{r},\mathtt{a}) \;\mid\; \exists T' \colon T \xrightarrow{\neq(\mathtt{s}:\,\mathsf{do}^{u})}{}^{*}\, T' \;\wedge\; (\mathtt{r},\mathtt{a}) \in \mathtt{Prs}^{u}_{\mathtt{s}}\big(T'\big) \right\}$$

In Definition 5.6, we are not interested in the actions that do not relate to the session u. Thus, we allow the network to evolve either by (*i*) letting any other participant other than $\mathtt{s}$ do an action, or (*ii*) letting $\mathtt{s}$ act on a different session than u, or (*iii*) do internal actions.

We now introduce the final ingredient for honesty, that is the notion of *readiness* of a participant.

Definition 5.7 (Readiness)**.** *We say that* $\mathtt{s}$ *is ready in* T *iff, whenever* $T \equiv (\vec{u},\vec{\mathbf{n}})T_1 \mid T_2$ *for*

some $\vec{u}, \vec{\mathbf{n}}$ *and* $T_1 = k[\![\mathtt{s}[P] \mid \ldots]\!] \mid T_0$*, the following holds:*

$$\exists \mathcal{X} \in \mathtt{Crs}(P) : \Big((\mathtt{r}, \mathtt{a}) \in \mathcal{X} \implies (\mathtt{r}, \mathtt{a}) \in \mathtt{WPrs}^k_{\mathtt{s}}(T_1) \Big)$$

Definition 5.7 says that a participant s is *ready* in a network T whenever its process ready set for a session k will eventually contain all the participant/sort pairs of a set in the contract ready set of s's contract in k. When a participant s is "ready", then, for any of its contracts P, the CO_2 process of s is (eventually) able to fulfil at least the interactions in P's prefix.

Remark 5.4. *The side condition "*$u \notin \vec{v}$*" of Definition 5.5 deals with cases like*

$$T_0 = (k)\left(\mathtt{s}\left[\!\left[\mathsf{do}^k_{\mathtt{r}}\,\mathsf{int}\right]\!\right]\right) \quad \textit{and} \quad T = T_0 \mid k[\![\mathtt{s}[\mathtt{r}!\mathsf{int}] \mid \ldots]\!] \mid \ldots$$

without the side condition, $\mathtt{Prs}^u_{\mathtt{k}}(T_0) = \{\{(\mathtt{r}, \mathsf{int})\}\}$*; thus, by Definition 5.7,* s *would result to be ready in* T*.*

Example 5.9. We have seen that, after fusion of the latent contracts of $T_{\mathtt{s}\mathtt{b}_1\mathtt{b}_2}$(in Example 5.3) we obtain:

$$(k)T'_{\mathtt{s}\mathtt{b}_1\mathtt{b}_2} \quad = \quad (k)\big(\mathtt{s}[\![R_{\mathtt{s}}\sigma\pi]\!] \;\mid\; k[\![S_{\mathtt{s}\mathtt{b}_1\mathtt{b}_2} \mid \mathtt{Q}(S_{\mathtt{s}\mathtt{b}_1\mathtt{b}_2})]\!] \;\mid\; \mathtt{b}_1[\![R_{\mathtt{b}_1}\sigma\pi]\!] \;\mid\; \mathtt{b}_2[\![R_{\mathtt{b}_2}\sigma\pi]\!]\big)$$

Let us define the processes (after substitutions):

$$\begin{aligned}
R_{\mathtt{s}}\sigma\pi &= \mathsf{do}^k_{\mathtt{b}_1}\,\mathsf{req} \,.\, \mathsf{do}^k_{\mathtt{b}_2}\,\mathsf{req} \,.\, \mathsf{do}^k_{\mathtt{b}_1}\,\mathsf{quote} \,.\, \big(\mathsf{do}^k_{\mathtt{b}_1}\,\mathsf{order} \,.\, \mathsf{do}^k_{\mathtt{b}_2}\,\mathsf{ok} \;+\; \mathsf{do}^k_{\mathtt{b}_1}\,\mathsf{bye} \,.\, \mathsf{do}^k_{\mathtt{b}_2}\,\mathsf{bye}\big) \\
R_{\mathtt{b}_1}\sigma\pi &= \tau \,.\, \mathsf{do}^k_{\mathtt{s}}\,\mathsf{req} \,.\, \mathsf{do}^k_{\mathtt{s}}\,\mathsf{quote} \,.\, \mathsf{do}^k_{\mathtt{s}}\,\mathsf{order} \\
R_{\mathtt{b}_2}\sigma\pi &= \mathsf{do}^k_{\mathtt{s}}\,\mathsf{req} \,.\, \big(\mathsf{do}^k_{\mathtt{b}_1}\,\mathsf{ok} \,.\, \mathsf{do}^k_{\mathtt{s}}\,\mathsf{ok} \;+\; \mathsf{do}^k_{\mathtt{b}_1}\,\mathsf{bye} \,.\, \mathsf{do}^k_{\mathtt{s}}\,\mathsf{bye}\big)
\end{aligned}$$

Thus, we have:

$$\begin{array}{lcccl}
\mathtt{Prs}^k_{\mathtt{s}}\left(T'_{\mathtt{sb_1b_2}}\right) & = & \{(\mathtt{b_1}, \mathsf{req})\} & = & \mathtt{WPrs}^k_{\mathtt{s}}\left(T'_{\mathtt{sb_1b_2}}\right) \\
\mathtt{Prs}^k_{\mathtt{b_1}}\left(T'_{\mathtt{sb_1b_2}}\right) & = & \varnothing \neq \{(\mathtt{s}, \mathsf{req})\} & = & \mathtt{WPrs}^k_{\mathtt{b_1}}\left(T'_{\mathtt{sb_1b_2}}\right) \\
\mathtt{Prs}^k_{\mathtt{b_2}}\left(T'_{\mathtt{sb_1b_2}}\right) & = & \{(\mathtt{s}, \mathsf{req})\} & = & \mathtt{WPrs}^k_{\mathtt{b_2}}\left(T'_{\mathtt{sb_1b_2}}\right)
\end{array}$$

Note that the τ prefix in $P_{\mathtt{b_1}}$ prevents $\mathtt{b_1}$ from interacting immediately with $\mathtt{s}$ on session k, although it is "weakly ready" to do so. Hence, considering that each weak process ready set of each participant in $T'_{\mathtt{sb_1b_2}}$ matches their respective contract ready set in $S_{\mathtt{sb_1b_2}}$ (Example 5.8) according to Definition 5.7 we have that participants $\mathtt{s}$, $\mathtt{b_1}$ and $\mathtt{b_2}$ are all ready in $(k)T'_{\mathtt{sb_1b_2}}$. ◇

Definition 5.8 (Honesty)**.** *We say that* $\mathtt{s}[\![R]\!]$ *is* honest *iff, for all* T *such that neither latent/stipulated contracts of* $\mathtt{s}$ *nor* $\mathtt{s}[\![\ldots]\!]$ *occur in* T*, and for all* T' *such that* $\mathtt{s}[\![R]\!] \mid T \rightarrow^* T'$*,* $\mathtt{s}$ *is ready in* T'*.*

A process $\mathtt{s}[\![R]\!]$ is said to be honest when, for all contexts and reductions that $\mathtt{s}[\![R]\!]$ may be engaged in, $\mathtt{s}$ is persistently ready. In other words, there is a continuous correspondence between the interactions exposed in the contract ready sets and the process ready sets of the possible reductions of any network involving $\mathtt{s}[\![R]\!]$. The definition rules out contexts with latent/stipulated contracts of $\mathtt{s}$, otherwise $\mathtt{s}$ could be made trivially dishonest, e.g., by inserting a latent contract $\downarrow_u \mathtt{s}\,\mathsf{says}\,P$ that $\mathtt{s}$ cannot fulfil.

Similarly to [4], the definition of honesty assumes a *fair* scheduler, eventually allowing participants to fire persistently (weakly) enabled do actions.

Example 5.10. Consider the process $\mathtt{b_1}[\![\mathsf{tell}_{\mathtt{s}} \downarrow_y P_{\mathtt{b_1}} \,.\, R_{\mathtt{b_1}}]\!]$ of network $T_{\mathtt{sb_1b_2}}$, as defined in Examples 5.3 and 5.9. We show that this process is *not* honest. In fact, $T_{\mathtt{sb_1b_2}}$ reduces as follows:

$$T_{\mathtt{sb_1b_2}} \rightarrow^* (k)T'_{\mathtt{sb_1b_2}} \rightarrow^* (k)T''_{\mathtt{sb_1b_2}}$$

where:

$$
\begin{aligned}
(k)T''_{\mathtt{s}\mathtt{b}_1\mathtt{b}_2} \;=\; (k)\Big(\; & \mathtt{s}\left[\!\left[\mathsf{do}^k_{\mathtt{b}_1}\,\mathsf{order}\,.\,\mathsf{do}^k_{\mathtt{b}_2}\,\mathsf{ok} \;+\; \mathsf{do}^k_{\mathtt{b}_1}\,\mathsf{bye}\,.\,\mathsf{do}^k_{\mathtt{b}_2}\,\mathsf{bye} \right]\!\right] \\
& \mid\; k\left[\!\left[\; \mathtt{s}[\mathtt{b}_1?\mathsf{order};\mathtt{b}_2!\mathsf{ok} \;+\; \mathtt{b}_1?\mathsf{bye};\mathtt{b}_2!\mathsf{bye}] \right.\right. \\
& \qquad \mid \mathtt{b}_1[\mathtt{b}_2!\mathsf{ok};\mathtt{s}!\mathsf{order} \oplus \mathtt{b}_2!\mathsf{bye};\mathtt{s}!\mathsf{bye}] \\
& \qquad \mid \mathtt{b}_2[\mathtt{b}_1?\mathsf{ok};\mathtt{s}?\mathsf{ok} \;+\; \mathtt{b}_1?\mathsf{bye};\mathtt{s}?\mathsf{bye}] \\
& \qquad \left.\left. \mid \mathtt{s}\mathtt{b}_1 : \varepsilon \mid \mathtt{b}_1\mathtt{s} : \varepsilon \mid \mathtt{s}\mathtt{b}_2 : \varepsilon \mid \mathtt{b}_2\mathtt{s} : \varepsilon \mid \mathtt{b}_1\mathtt{b}_2 : \varepsilon \mid \mathtt{b}_2\mathtt{b}_1 : \varepsilon \;\right]\!\right] \\
& \mid\; \mathtt{b}_1\left[\!\left[\mathsf{do}^k_{\mathtt{s}}\,\mathsf{order} \right]\!\right] \;\mid\; \mathtt{b}_2\left[\!\left[\mathsf{do}^k_{\mathtt{b}_1}\,\mathsf{ok}\,.\,\mathsf{do}^k_{\mathtt{s}}\,\mathsf{ok} \;+\; \mathsf{do}^k_{\mathtt{b}_1}\,\mathsf{bye}\,.\,\mathsf{do}^k_{\mathtt{s}}\,\mathsf{bye} \right]\!\right] \Big)
\end{aligned}
$$

At this point, we notice a problem in the implementation of $\mathtt{b}_1$: it does not notify the other buyer before making an order.

In fact, $\mathtt{b}_1$'s process is trying to perform $\mathsf{do}^k_{\mathtt{s}}\,\mathsf{order}$, but its contract requires that $\mathsf{do}^k_{\mathtt{b}_2}\,\mathsf{ok}$ is performed first (or $\mathsf{do}^k_{\mathtt{b}_2}\,\mathsf{bye}$, if the quote is rejected). This is reflected by the mismatch between $\mathtt{b}_1$'s process ready set in $T''_{\mathtt{s}\mathtt{b}_1\mathtt{b}_2}$ and its contract ready sets, in session k:

$$
\begin{aligned}
\mathtt{Prs}^k_{\mathtt{b}_1}\left(T''_{\mathtt{s}\mathtt{b}_1\mathtt{b}_2}\right) &= \{\{(\mathtt{s},\mathsf{order})\}\} \\
\mathtt{Crs}(\mathtt{b}_2!\mathsf{ok};\mathtt{s}!\mathsf{order} \oplus \mathtt{b}_2!\mathsf{bye};\mathtt{s}!\mathsf{bye}) &= \{\{(\mathtt{b}_2,\mathsf{ok})\},\{(\mathtt{b}_2,\mathsf{bye})\}\}
\end{aligned}
$$

Using the vocabulary of Definitions 5.3, 5.7, and 5.8, we have:

- There exists a network $T_{\mathtt{s}\mathtt{b}_1\mathtt{b}_2}$ which contains $\mathtt{b}_1[\![\mathsf{tell}_{\mathtt{s}} \downarrow_y P_{\mathtt{b}_1}\,.\,R_{\mathtt{b}_1}]\!]$
- and reduces to a network $(k)T''_{\mathtt{s}\mathtt{b}_1\mathtt{b}_2}$, where $\mathtt{b}_1$ is *not ready*.
- Thus, we have that $\mathtt{b}_1[\![\mathsf{tell}_{\mathtt{s}} \downarrow_y P_{\mathtt{b}_1}\,.\,R_{\mathtt{b}_1}]\!]$ is not honest, and
- $\mathtt{b}_1$ is *culpable* in $(k)T''_{\mathtt{s}\mathtt{b}_1\mathtt{b}_2}$.

◇

Remark 5.5. *In this section, we consider only fully instantiated contracts, i.e., contracts that do not use participant variables. This is due to two reasons. (i) Honesty is defined on*

all the possible executions of any networks, thus it encompasses any possible instantiation of participant variables. (ii) The results of Section 5.6 apply only whenever sessions have been started, which implies that participant variables have substituted by participant names.

Remark 5.6. *Honesty is not decidable in general [7], but for a bilateral contract model it has been approximated via an abstract semantics [7] and a type discipline [4] for CO_2. We conjecture that these approximations may be adapted to our setting.*

5.6 Properties of Honest Networks

We give the properties that our framework guarantees. We ensure that two basic features of CO_2 hold in our multiparty adaptation: the state of a session always permits to establish who is responsible for making the network progress (Theorem 5.1) and honest participants can always exculpate themselves (Lemma 5.1). We then formalise a link between the honesty of participants and two key properties borrowed from the session types world: Theorem 5.2 introduces session fidelity in CO_2; and Theorem 5.3 introduces a notion of progress in CO_2, based on the progress of the contractual agreement.

First, we define the kind of networks on which our results apply, i.e., *honest networks*. These networks consists only of honest participants and do not feature any runtime constructs (such as sessions or stipulated contracts).

Definition 5.9 (Initial & honest networks)**.** *A network T is* initial *if*

- *for each participant* $\mathtt{s}$ *in T, there is at most one sub-network* $\mathtt{s}[\![Z]\!]$ *in T,*
- *for each sub-network* $\mathtt{s}[\![Z]\!]$ *in T, there is no (latent or stipulated) contract in Z that does not belong to* $\mathtt{s}$*, i.e., Z has no sub-term of the form* $\mathsf{tell}_{\underline{\mathtt{n}}}\downarrow_u P$ *or* $\downarrow_u \mathtt{n}\,\mathsf{says}\,P$*, with* $\mathtt{n} \neq \mathtt{s}$*, and*

- *no session has been started in T, i.e., there is no sub-network of the form $k[\![S]\!]$ in T.*

An honest network T *is an* initial *network such that each process* $\mathtt{s}[\![R]\!]$ *in* T *is honest.*

Theorem 5.1 below says that, in an active session, there is always at least one participant $\mathtt{s}[\![R]\!]$ who is responsible for the next interaction. Thus, if a corresponding $\mathsf{do}^{k}_{\mathtt{r}}\,\mathsf{a}$ prefix is not in R, T may get stuck, and $\mathtt{s}$ is culpable.

Theorem 5.1 (Culpability)**.** *Given a network T, if T contains a session $k[\![S]\!]$ such that $\exists\,\mathtt{n} \in \mathcal{P}(S) :\ S(\mathtt{n}) \neq \mathbf{0}$, then there exists at least one* culpable *participant.*

Proof. Since session k is not terminated, we must have:

$$S \equiv \mathtt{n}[\mathtt{r}!\mathsf{a}\,;P \oplus P] \mid S' \quad \text{or} \quad S \equiv \mathtt{n}[\mathtt{s}?\mathsf{a}\,;P + P] \mid \mathtt{sn} : \mathsf{a} \cdot \rho \mid S'$$

Note that, by Theorem 3.1, S cannot be a deadlock or reduce to an orphan message configuration.

Since k has been started (cf. Remark 5.2) and $\mathtt{n}$ is part of the session, we must have T of the form

$$T \equiv (\vec{u}, \vec{\mathbf{n}})\,(\mathtt{n}[\![R]\!] \mid k[\![S]\!] \mid T_1) \mid T_2$$

with $k \in \vec{u}$, $\vec{\mathbf{n}}$ possibly empty, and possibly $R \equiv \mathbf{0}$, note that we have $\mathtt{n}[\![\mathbf{0}]\!] \equiv \mathbf{0}$ from the congruence rules of CO_2. Thus, we have $S \xrightarrow{\mathtt{n}\to\mathtt{r}:\mathsf{a}}$ or $S \xrightarrow{\mathtt{n}\leftarrow\mathtt{s}:\mathsf{a}}$ as required by Definition 5.3. □

Example 5.11. Consider network $T''_{\mathtt{sb}_1\mathtt{b}_2}$ of Example 5.10 and the system of contracts in its session k:

$$\begin{aligned} S_k \;=\;& \mathtt{s}[\mathtt{b}_1?\mathsf{order}\,;\mathtt{b}_2!\mathsf{ok} \;+\; \mathtt{b}_1?\mathsf{bye}\,;\mathtt{b}_2!\mathsf{bye}] \\ & \mid \mathtt{b}_1[\mathtt{b}_2!\mathsf{ok}\,;\mathtt{s}!\mathsf{order} \oplus \mathtt{b}_2!\mathsf{bye}\,;\mathtt{s}!\mathsf{bye}] \;\mid\; \mathtt{b}_2[\mathtt{b}_1?\mathsf{ok}\,;\mathtt{s}?\mathsf{ok} \;+\; \mathtt{b}_1?\mathsf{bye}\,;\mathtt{s}?\mathsf{bye}] \\ & \mid \mathtt{sb}_1 : \varepsilon \mid \mathtt{b}_1\mathtt{s} : \varepsilon \mid \mathtt{sb}_2 : \varepsilon \mid \mathtt{b}_2\mathtt{s} : \varepsilon \mid \mathtt{b}_1\mathtt{b}_2 : \varepsilon \mid \mathtt{b}_2\mathtt{b}_1 : \varepsilon \end{aligned}$$

We have $S_k \xrightarrow{\mathtt{b}_1 \leftrightharpoons \mathtt{b}_2 : \mathsf{ok}}$ and $S_k \xrightarrow{\mathtt{b}_1 \leftrightharpoons \mathtt{b}_2 : \mathsf{bye}}$. Hence, $\mathtt{b}_1$ is responsible for the next interaction, and culpable for $T''_{\mathtt{s}\mathtt{b}_1\mathtt{b}_2}$ being stuck. ◇

Lemma 5.1 follows from the definition of honesty. It states that honest participants can always overcome their culpability, by firing their contractual do actions (possibly after some internal actions).

Lemma 5.1 (Exculpation)**.** *Given an initial network T_0 with an honest participant $\mathtt{s}[\![R]\!]$, whenever $T_0 \rightarrow^* T \equiv (\vec{u}, \vec{\mathbf{n}})\,(k[\![S]\!] \mid T_1) \mid T_2$ and $\mathtt{s}$ is culpable in T, there exist $\mathtt{r}$ and $\mathtt{a}$ such that:*

$$T \xrightarrow{\mathtt{s} : \tau}{}^{*} \xrightarrow{\mathtt{s} : \mathsf{do}^k_{\mathtt{r}}\,\mathtt{a}}$$

Proof. By Definition 5.8, we have that for any reduction of T_0, $\mathtt{s}$ is *ready*. By assumption, $\mathtt{s}$ is culpable in T, so we must have $S \xrightarrow{\mathtt{s} \leftrightharpoons \mathtt{r} : \mathtt{a}}$, and, by Definition 5.7, we must have $(\mathtt{r}, \mathtt{a}) \in \mathtt{WPrs}^k_{\mathtt{s}}(T_1)$. By Definition 5.6, we have

$$\exists T' \colon T_1 \xrightarrow{\neq(\mathtt{s} : \mathsf{do}^k)}{}^{*} T' \;\wedge\; (\mathtt{r}, \mathtt{a}) \in \mathtt{Prs}^k_{\mathtt{s}}(T')$$

By Definition 5.5, we must have that $\mathtt{s}\left[\!\!\left[\mathsf{do}^k_{\mathtt{r}}\,\mathtt{a}\,.\,R\right]\!\!\right]$ is a sub-network of T'.[3] Thus, we just have to show that $\mathtt{s}$ may reduce to such a prefixed process via internal actions (τ) *only*. By contradiction, if it was the case that, in the process belonging to $\mathtt{s}$, there was a blocking action (before $\mathsf{do}^k_{\mathtt{r}}\,\mathtt{a}$), then one could easily find a network in which $\mathtt{s}$ is not ready (i.e., $\mathtt{s}$ would be waiting for this action to be dealt with by another participant); thus contradicting the assumption that $\mathtt{s}$ is honest. □

Theorem 5.2 below says that each (honest) participant will strictly adhere to its contracts, once they have been fused in a session. It follows directly from the semantics of CO_2 (that forbid non-contractual do prefixes to be fired) and from the definition of honesty.

[3] We abstract from possible choice or variable restrictions, without loss of generality.

Theorem 5.2 (Fidelity)**.** *Let T be an honest network, if*

$$T \to^* T' \equiv (\vec{u}, \vec{\mathtt{n}})\,(\mathtt{s}[\![R]\!] \mid k[\![S]\!] \mid T_0) \mid T_1$$

then

$$T' \xrightarrow{-k}{}^* \xrightarrow{\mathtt{s}\,:\,\mathsf{do}^k_{\mathtt{r}}\mathsf{a}} \quad \iff \quad S \xrightarrow{\mathtt{s}\leftrightharpoons\mathtt{r}:\mathsf{a}}$$

(where $\xrightarrow{-k}{}^*$ is any reduction not *involving session k).*

Proof. The direction ($\Rightarrow$) follows directly from the semantics of CO_2 (i.e., rule [DO]). The direction ($\Leftarrow$) follows from Lemma 5.1. Indeed, $S \xrightarrow{\mathtt{s}\leftrightharpoons\mathtt{r}:\mathsf{a}}$ implies that $\mathtt{s}$ is culpable in session k, and by Lemma 5.1, $\mathtt{s}$ is able to exculpate itself after some internal actions (i.e., no actions on session k). □

Theorem 5.3 below introduces a notion of global progress in CO_2 networks. Observe that progress in CO_2 is only meaningful *after* a session has been established and thus a culpable participant exists. A network without sessions may not progress because a set of compliant contracts cannot be found, or a fuse prefix is not enabled. In both cases, no participant may be deemed culpable, and thus responsible for the next move. However, the network may progress again if other (honest) participants join it, allowing a session to be established. This is analogous to the notion of progress of [35], cf. Section 2.2.3.

Theorem 5.3 (Global progress)**.** *Given an honest network T_0, whenever*

$$T_0 \to^* T \equiv (\vec{u}, \vec{\mathtt{n}})\,(k[\![S]\!] \mid T_1) \mid T_2$$

and $\exists\, \mathtt{n} \in \mathcal{P}(S) :\ S(\mathtt{n}) \neq \mathbf{0}$, then $T \not\to$.

Proof. By contradiction. By Theorem 3.1, we know that S is deadlock free, thus if the network cannot make further reductions, it must be because one of the participant $\mathtt{n}$ cannot

meet its obligations. This is a contradiction with Lemma 5.1 since all participants are honest in T_0. □

Note that Theorem 5.3 holds for networks where a process takes part in multiple sessions: the honesty of all participants guarantees that all sessions will be completed. We illustrate such a situation in Example 5.12 below.

Example 5.12. We show how a seemingly honest process (r) could be deemed culpable due to the unexpected behaviour of other participants. Consider the following network:

$$\begin{aligned} T_{\text{ex5.12}} \quad = \quad (x,y,z,w)\big(\quad & \mathtt{s}[\![\mathsf{tell}_{\mathtt{s}} \downarrow_x (\mathtt{r}!\mathsf{int}) \,.\, \mathsf{fuse} \,.\, \mathsf{fuse}]\!] \\ | \quad & \mathtt{r}[\![\mathsf{tell}_{\mathtt{s}} \downarrow_y (\mathtt{s}?\mathsf{int}) \,.\, \mathsf{tell}_{\mathtt{s}} \downarrow_z (\mathtt{n}!\mathsf{bool}) \,.\, \mathsf{do}^{y}_{\mathtt{s}}\, \mathsf{int} \,.\, \mathsf{do}^{z}_{\mathtt{n}}\, \mathsf{bool}]\!] \\ | \quad & \mathtt{n}[\![\mathsf{tell}_{\mathtt{s}} \downarrow_w (\mathtt{r}?\mathsf{bool}) \,.\, \mathsf{do}^{w}_{\mathtt{r}}\, \mathsf{bool}]\!] \,\big) \end{aligned}$$

where participant s advertises a contract to itself (promising to sent a message int to r); participant r advertises two contracts to s, the first contract specifies an interaction with s, while the second specifies an interaction with n; and participant n advertises a contract (to s) specifying an interaction with r.

After all four contracts have been advertised to s and fused, the network reduces to:

$$\begin{aligned} T'_{\text{ex5.12}} = (k_1,k_2)\big(\; \mathtt{s}[\![\mathbf{0}]\!] \quad | \quad & \mathtt{r}\left[\!\!\left[\mathsf{do}^{k_1}_{\mathtt{s}}\, \mathsf{int} \,.\, \mathsf{do}^{k_2}_{\mathtt{n}}\, \mathsf{bool}\right]\!\!\right] \quad | \quad \mathtt{n}\left[\!\!\left[\mathsf{do}^{k_2}_{\mathtt{r}}\, \mathsf{bool}\right]\!\!\right] \\ | \quad & k_1[\![\mathtt{s}[\mathtt{r}?\mathsf{int}] \,|\, \mathtt{r}[\mathtt{s}!\mathsf{int}] \,|\, \mathtt{sr} : \varepsilon \,|\, \mathtt{rs} : \varepsilon]\!] \\ | \quad & k_2[\![\mathtt{r}[\mathtt{n}!\mathsf{bool}] \,|\, \mathtt{n}[\mathtt{r}?\mathsf{bool}] \,|\, \mathtt{rn} : \varepsilon \,|\, \mathtt{nr} : \varepsilon]\!] \,\big) \end{aligned}$$

Even if the systems of contracts in both sessions k_1 and k_2 are able to reduce further, $T'_{\text{ex5.12}}$ is stuck. Indeed, s does not perform the promised action, it is culpable in k_1; and r is stuck waiting in k_1, thus remaining culpable[4] in k_2. Indeed, neither s nor r are *ready* in $T'_{\text{ex5.12}}$, and thus they are not honest in $T_{\text{ex5.12}}$. Hence, the network cannot progress

[4] In this case, r is deemed culpable in k_2 because its implementation did not expect s to misbehave.

further. ◇

Example 5.13. Let us now consider a variant of $T_{\text{ex5.12}}$ from Example 5.12, where all participants are honest:

$$
\begin{aligned}
(x,y,z,w)\big(\ & \mathtt{s}[\![(\mathsf{tell}_{\mathtt{s}} \downarrow_x (\mathtt{r}!\mathsf{int}) \,.\, \mathsf{do}^x_{\mathtt{r}}\,\mathsf{int}) \mid \mathsf{fuse} \mid \mathsf{fuse}]\!] \\
& \mid \mathtt{n}[\![\mathsf{tell}_{\mathtt{s}} \downarrow_w (\mathtt{r}?\mathsf{bool}) \,.\, \mathsf{do}^w_{\mathtt{r}}\,\mathsf{bool}]\!] \\
& \mid \mathtt{r}[\![\mathsf{tell}_{\mathtt{s}} \downarrow_y (\mathtt{s}?\mathsf{int}) \,.\, \mathsf{tell}_{\mathtt{s}} \downarrow_z (\mathtt{n}!\mathsf{bool}) \,.\, \big(\mathsf{do}^y_{\mathtt{s}}\,\mathsf{int} \,.\, \mathsf{do}^z_{\mathtt{n}}\,\mathsf{bool} + \tau \,.\, (\mathsf{do}^y_{\mathtt{s}}\,\mathsf{int} \mid \mathsf{do}^z_{\mathtt{n}}\,\mathsf{bool})\big)]\!]\ \big)
\end{aligned}
$$

In this case, s will respect its contractual duties, while r will be ready to fulfil its contracts on both sessions – even if one is not activated, or remains stuck (here, τ represents an internal action, e.g., a timeout: if the first $\mathsf{do}^y_{\mathtt{s}}\,\mathsf{int}$ cannot reduce, r falls back to running the sessions in parallel). The honesty of all participants in the network above guarantees that, once a session is active, it will reach its completion. ◇

5.7 Concluding Remarks

In this chapter, we investigated the combination of the contract-oriented calculus CO_2 with a contract model that fulfils two basic design requirements: (i) it supports multiparty agreements, and (ii) it provides an explicit description of the choreography that embodies each agreement. To the best of our knowledge, no other contract model provides an explicit choreography synthesis.

We built our framework upon a simple version of session types and yet it turns out to be quite flexible, e.g., allowing for sessions where the number of participants is not known beforehand and permitting fine-grained characterisation of the contractual agreement.

We are considering two main directions of future research based on this work. The first direction concerns an extension of the fuse primitive so to allow that, whenever several sets of contracts are compliant, the "best" set of contracts is fused (according to

different criteria such as, e.g., the number of participants or a semantic characterisation of the choreography). The extensions we discussed in Section 5.4.2 relied on a non-deterministic selection of the set of contacts to be fused (via the $\equiv$ congruence relation). One may integrate the selection of contracts within the agreement relation so that the best contracts are chosen and the ones which are unused stay in the network, to be fused later on.

Another research direction concerns the possibility of allowing a participant to be involved in a same session with multiple contracts – e.g., a bank advertising two services, and a customer publishing a contract which uses both contracts of the bank in a same session. This is quite challenging as it may require to somehow merge these contracts into one, before attempting to synthesise a global type. Indeed, one may not be able to consider that the contracts are simply two different participants in the choreography, e.g., a process advertising several contracts may not fulfil them in an independent manner.

CHAPTER 6

Conclusions and Future Directions

6.1 Summary of the Contributions

We review the main contributions presented in Chapters 3, 4, and 5.

On synthesising choreographies. In Chapter 3, we introduced a theory whereby it is possible to infer a choreography from a set of local behavioural specifications. The main construction in this chapter is a type system which assigns a global type to a set of local types. We showed that, if it exists, such a global type is unique and well-formed; also, its projections are equivalent to the original local types. The type system enjoys a subject reduction property. Finally, we showed that for every well-formed global type, an equivalent global type can be assigned to its projections.

A major advantage of our bottom-up approach is that it allows the result of the top-down approach of [44] to be applicable even if no global view of the system is available. This allowed us to reuse well-established results to show that, if a system is typable, then it has progress and safety properties. As a consequence, our theory allows to lift the assumption that a global view of the system is available to type-check programs with

session types.

On amending global assertions. In Chapter 4, we gave a few techniques to help distributed software architects design global assertions. These techniques include two algorithms to solve history sensitivity problems, one algorithm to solve temporal satisfiability problems, and a methodology for applying the algorithms to protocol design.

We showed that our algorithms satisfy the following properties: (*i*) structure preservation, i.e., they do not modify the structure of the underlying global type, (*ii*) properties preservation, i.e., they do not introduce new violations, and (*iii*) correctness, i.e., if applicable they correct all the problems.

As the theory in Chapter 3 allows to lift the burden of (re-) designing global types, the methodology presented in Chapter 4 permits, to some extent, to lift the burden of making a global assertion well-asserted.

Runtime choreography synthesis. In Chapter 5, we introduced a formal model for distributed systems where participants advertise contracts and multiparty sessions are started when a set of contracts allows to synthesise a choreography.

Our model is based on an adaptation of the CO_2 calculus [6] and preserves its main properties. In particular, it is always possible to identify which participant is responsible for the session being stuck. Inspired by other results from session type theory, we also showed that progress and session fidelity properties hold for *honest* systems, i.e., systems consisting of honest participants only. In addition, we gave a few examples illustrating the utility of having a choreography as a tangible representation of an agreement between participants.

A remarkable feature of our model is that it adds flexibility to the synthesis of choreographies of Chapter 3 by allowing *participant variables* to appear in contracts.

6.2 Future Directions

We identify a few directions of future research based on the results established in this thesis. We focus on the synthesis of choreographies as the other topics where covered in the concluding remarks of their respective chapters.

From communicating machines to choreographies. A main direction of future work, which has been confirmed in preliminary results by the author and colleagues (and to some extent in [41]), is to study the synthesis of choreographies from communicating machines. Remarkably, such a synthesis allows to extend substantially the set of systems to which it is possible to assign a choreography. Indeed, contrarily to the type system of Chapter 3, there are fewer syntactic restrictions: this is most noticeable for recursive types. In fact, using the theory of regions [3, 37], it is possible to synthesise a choreography – or a generalised global types [40] (cf. Section 2.1.4) – from the *synchronous* transition system of a system of communicating machines. The synchronous transition system of a system of communicating machines amounts to, essentially, letting the machines run with the additional constraint that at most one buffer is non-empty, that the non-empty buffer contains at most one symbol, and that an output action is directly followed by its corresponding input action.

Since communicating machines are undecidable in general [16], one needs to restrict the set of "typable" systems. In fact, an important step is to check whether all the executions of the machines are "captured" by the synchronous transition system.

In addition, we conjecture that the results of Chapters 4 and 5 may be quite easily extended to generalised global types. On the one hand, the framework of Chapter 5 is in fact rather independent of the way choreographies are actually synthesised. It uses the type system of Chapter 3 as a "black box". On the other hand, in order to adapt the results of Chapter 4, one would need to re-visit the notion of well-assertedness for general graphs and similarly for the three algorithms. In fact, the main difference is that there might be

several paths to a same node. Thus, our algorithms would have to update all the paths leading to a problematic node.

Beyond 1-buffer executions. One of the main restriction of our type system and, to the best of our knowledge, all the current works on the synthesis of choreographies [41], is that they do not capture systems that have executions which are not captured by a synchronous execution. Analogously, the conditions for the *realisability* of choreography (cf. Section 2.2.1 and [9, 10]) also requires a system to be *synchronisable*. See Section 3.6 for a discussion on the 1-buffer restriction in our type system.

Example 6.1 below shows that some systems cannot be assigned a choreography in any of the works we discussed in Chapter 2.

Example 6.1. Consider the following system of two local types:

$$\mathtt{s}[\mathtt{r}!\mathtt{a};\mathtt{r}?\mathtt{b}] \mid \mathtt{r}[\mathtt{s}!\mathtt{b};\mathtt{s}?\mathtt{a}] \tag{6.1}$$

it is easy to see that this system does not deadlock, however it is not typable by the type system of Chapter 3.[1] Essentially, this is due to the fact that the system "deadlocks" when each intermediary configuration may contain at most one non-empty buffer. Indeed, with such a constraint the system reduces, e.g, to

$$\mathtt{s}[\mathtt{r}?\mathtt{b}] \mid \mathtt{r}[\mathtt{s}!\mathtt{b};\mathtt{s}?\mathtt{a}] \mid \mathtt{sr} : \mathtt{a} \mid \mathtt{rs} : \varepsilon$$

where r cannot send b since one buffer is already non empty and a cannot be received since r must send b first.

In fact, the system (6.1) cannot be captured by any of the formalism we have seen so

[1] This system is also not *multiparty-compatible* in the sense of [41].

far. For instance, neither of these global types capture it:

$$\mathtt{s}\to\mathtt{r}:\mathtt{a};\ \mathtt{r}\to\mathtt{s}:\mathtt{b}\quad\text{✘}\qquad\qquad \mathtt{r}\to\mathtt{s}:\mathtt{b};\ \mathtt{s}\to\mathtt{r}:\mathtt{a}\quad\text{✘}$$

Even if we allow a same participant to appear in different concurrent branches and introduce parallel constructs at the local type level, projecting the global type

$$\mathtt{s}\to\mathtt{r}:\mathtt{a}\ \mid\ \mathtt{r}\to\mathtt{s}:\mathtt{b}$$

would yield the system:

$$\mathtt{s}[\mathtt{r}!\mathtt{a}\mid\mathtt{r}?\mathtt{b}]\ \mid\mathtt{r}[\mathtt{s}!\mathtt{b}\mid\mathtt{s}?\mathtt{a}] \tag{6.2}$$

which is not bisimilar to system (6.1). Observe that the system (6.2) is not directly representable in terms of communicating machines since CFSMs do not feature (explicit) concurrent activities. ◇

Example 6.1 shows that taking asynchrony to its full potential when synthesising a choreography is not easy. In fact, choreography languages consider interactions of the form $\mathtt{s}\to\mathtt{r}:\mathtt{a}$ to be *atomic*; and they generally offer three main constructs: sequence, parallel, and choice. None of these constructs allow to model system (6.1) from a global perspective. In fact, before attempting to tackle the problem of synthesising a choreography from systems that do not have a synchronous execution, one would need to consider a choreography language and a notion of projection that do cater for (explicit) asynchrony. For instance, one may consider a notion of projection that gives precedence to sending actions in concurrent branches so that the projection of

$$\mathtt{s}\to\mathtt{r}:\mathtt{a}\ \mid\ \mathtt{r}\to\mathtt{s}:\mathtt{b}$$

yields

$$\mathtt{s}[\mathtt{r}!\mathtt{a};\mathtt{r}?\mathtt{b}] \mid \mathtt{r}[\mathtt{s}!\mathtt{b};\mathtt{s}?\mathtt{a}]$$

Other directions. Other directions of research concerns a more precise comparison between our type system and other related approaches, such as the conversation types [23] and the work of Castagna et al. [31], so to characterise the systems that are supported by each theory.

Another interesting direction is to work on a methodology that combines our bottom-up approach with top-down approaches such as "global programming" [26]. The objective being that both directions, from global to local specifications, and vice versa, should be available to practitioners. Combining both approaches would not only help to keep specifications and implementations as close as possible to each other, but also facilitate the adoption of session type theory in more ad-hoc development cycles. Similarly to what is done in [1], our bottom-up approach may also be valuable to infer "bad scenarios" so that practitioners may more easily understand why an implementation does not match a given choreography.

Finally, we are currently working on a tool that constructs a choreography (in a syntax similar to the one of generalised global types) from a set of communicating machines, based on the preliminary results mentioned above. We are also considering implementing both the type system of Chapter 3 and the algorithms of Chapter 4, so to integrate them in tool described in [52].

Bibliography

[1] Rajeev Alur, Kousha Etessami, and Mihalis Yannakakis. Inference of message sequence charts. *IEEE Trans. Software Eng.*, 29(7):623–633, 2003.

[2] Krzysztof R. Apt, Nissim Francez, and Shmuel Katz. Appraising fairness in languages for distributed programming. *Distributed Computing*, 2(4):226–241, 1988.

[3] Eric Badouel and Philippe Darondeau. Theory of regions. In Wolfgang Reisig and Grzegorz Rozenberg, editors, *Petri Nets*, volume 1491 of *Lecture Notes in Computer Science*, pages 529–586. Springer, 1996.

[4] Massimo Bartoletti, Alceste Scalas, Emilio Tuosto, and Roberto Zunino. Honesty by typing. In Dirk Beyer and Michele Boreale, editors, *FMOODS/FORTE*, volume 7892 of *Lecture Notes in Computer Science*, pages 305–320. Springer, 2013.

[5] Massimo Bartoletti, Emilio Tuosto, and Roberto Zunino. Contracts in distributed systems. In Silva et al. [66], pages 130–147.

[6] Massimo Bartoletti, Emilio Tuosto, and Roberto Zunino. Contract-oriented computing in CO_2. *Scientific Annals in Comp. Sci.*, 22(1):5–60, 2012.

[7] Massimo Bartoletti, Emilio Tuosto, and Roberto Zunino. On the realizability of

contracts in dishonest systems. In Marjan Sirjani, editor, *COORDINATION*, volume 7274 of *Lecture Notes in Computer Science*. Springer, 2012.

[8] Massimo Bartoletti and Roberto Zunino. A calculus of contracting processes. In *LICS*. IEEE Computer Society, 2010.

[9] Samik Basu, Tevfik Bultan, and Meriem Ouederni. Deciding choreography realizability. In John Field and Michael Hicks, editors, *POPL*, pages 191–202. ACM, 2012.

[10] Samik Basu, Tevfik Bultan, and Meriem Ouederni. Synchronizability for verification of asynchronously communicating systems. In Viktor Kuncak and Andrey Rybalchenko, editors, *VMCAI*, volume 7148 of *LNCS*. Springer, 2012.

[11] Lorenzo Bettini, Mario Coppo, Loris D'Antoni, Marco De Luca, Mariangiola Dezani-Ciancaglini, and Nobuko Yoshida. Global progress in dynamically interleaved multiparty sessions. In Franck van Breugel and Marsha Chechik, editors, *CONCUR*, volume 5201 of *Lecture Notes in Computer Science*, pages 418–433. Springer, 2008.

[12] Laura Bocchi, Kohei Honda, Emilio Tuosto, and Nobuko Yoshida. A theory of design-by-contract for distributed multiparty interactions. In Gastin and Laroussinie [42], pages 162–176.

[13] Laura Bocchi, Julien Lange, and Emilio Tuosto. Amending contracts for choreographies. In Silva et al. [66], pages 111–129.

[14] Laura Bocchi, Julien Lange, and Emilio Tuosto. Three algorithms and a methodology for amending contracts for choreographies. *Scientific Annals of Computer Science*, 22(1):61–104, 2012.

[15] Michele Boreale and Rocco De Nicola. Testing equivalence for mobile processes. *Inf. Comput.*, 120(2):279–303, 1995.

[16] Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *JACM*, 30(2):323–342, 1983.

[17] Mario Bravetti, Ivan Lanese, and Gianluigi Zavattaro. Contract-driven implementation of choreographies. In Christos Kaklamanis and Flemming Nielson, editors, *TGC*, volume 5474 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2008.

[18] Maria Grazia Buscemi and Ugo Montanari. Cc-pi: A constraint-based language for specifying service level agreements. In Rocco De Nicola, editor, *ESOP*, volume 4421 of *Lecture Notes in Computer Science*, pages 18–32. Springer, 2007.

[19] Luís Caires and Frank Pfenning. Session types as intuitionistic linear propositions. In Gastin and Laroussinie [42], pages 222–236.

[20] Luís Caires, Frank Pfenning, and Bernardo Toninho. Towards concurrent type theory. In Benjamin C. Pierce, editor, *TLDI*, pages 1–12. ACM, 2012.

[21] Luís Caires and Hugo Torres Vieira. Conversation types. In Castagna [27], pages 285–300.

[22] Luís Caires and Hugo Torres Vieira. Analysis of service oriented software systems with the conversation calculus. In Luís Soares Barbosa and Markus Lumpe, editors, *FACS*, volume 6921 of *Lecture Notes in Computer Science*, pages 6–33. Springer, 2010.

[23] Luís Caires and Hugo Torres Vieira. Conversation types. *Theor. Comput. Sci.*, 411(51-52):4399–4440, 2010.

[24] Marco Carbone, Kohei Honda, and Nobuko Yoshida. Structured communication-centred programming for web services. In Rocco De Nicola, editor, *ESOP*, volume 4421 of *Lecture Notes in Computer Science*, pages 2–17. Springer, 2007.

[25] Marco Carbone, Kohei Honda, and Nobuko Yoshida. Structured communication-centered programming for web services. *ACM Trans. Program. Lang. Syst.*, 34(2):8, 2012.

[26] Marco Carbone and Fabrizio Montesi. Deadlock-freedom-by-design: multiparty asynchronous global programming. In Roberto Giacobazzi and Radhia Cousot, editors, *POPL*, pages 263–274. ACM, 2013.

[27] Giuseppe Castagna, editor. *Programming Languages and Systems, 18th European Symposium on Programming, ESOP 2009*, volume 5502 of *Lecture Notes in Computer Science*. Springer, 2009.

[28] Giuseppe Castagna, Mariangiola Dezani-Ciancaglini, and Luca Padovani. On global types and multi-party sessions. In Roberto Bruni and Jürgen Dingel, editors, *FMOODS/FORTE*, volume 6722 of *Lecture Notes in Computer Science*, pages 1–28. Springer, 2011.

[29] Giuseppe Castagna, Mariangiola Dezani-Ciancaglini, and Luca Padovani. On global types and multi-party session. *Logical Methods in Computer Science*, 8(1), 2012.

[30] Giuseppe Castagna, Nils Gesbert, and Luca Padovani. A theory of contracts for web services. *ACM Trans. on Prog. Lang. and Sys.*, 31(5), 2009.

[31] Giuseppe Castagna and Luca Padovani. Contracts for mobile processes. In Mario Bravetti and Gianluigi Zavattaro, editors, *CONCUR*, volume 5710 of *Lecture Notes in Computer Science*, pages 211–228. Springer, 2009.

[32] Tzu-Chun Chen, Laura Bocchi, Pierre-Malo Deniélou, Kohei Honda, and Nobuko Yoshida. Asynchronous distributed monitoring for multiparty session enforcement. In Roberto Bruni and Vladimiro Sassone, editors, *TGC*, volume 7173 of *Lecture Notes in Computer Science*, pages 25–45. Springer, 2011.

[33] Tzu-Chun Chen and Kohei Honda. Specifying stateful asynchronous properties for distributed programs. In Koutny and Ulidowski [49], pages 209–224.

[34] World Wide Web Consortium. Web services choreography description language version 1.0. `http://www.w3.org/TR/ws-cdl-10/`, 11 2005.

[35] Mario Coppo, Mariangiola Dezani-Ciancaglini, Luca Padovani, and Nobuko Yoshida. Inference of global progress properties for dynamically interleaved multiparty sessions. In Rocco De Nicola and Christine Julien, editors, *COORDINATION*, volume 7890 of *Lecture Notes in Computer Science*, pages 45–59. Springer, 2013.

[36] Mario Coppo, Mariangiola Dezani-Ciancaglini, Nobuko Yoshida, and Luca Padovani. Global progress for dynamically interleaved multiparty sessions. Technical report, Dipartimento di Informatica, Università di Torino, 2013. Available at `http://www.di.unito.it/~padovani/Papers/CoppoDezaniYoshidaPadovani13.pdf`.

[37] Jordi Cortadella, Michael Kishinevsky, Luciano Lavagno, and Alexandre Yakovlev. Deriving Petri Nets for Finite Transition Systems. *IEEE Trans. Computers*, 47(8):859–882, 1998.

[38] Gero Decker, Oliver Kopp, Frank Leymann, and Mathias Weske. Bpel4chor: Extending bpel for modeling choreographies. In *ICWS*, pages 296–303. IEEE Computer Society, 2007.

[39] Pierre-Malo Deniélou and Nobuko Yoshida. Dynamic multirole session types. In Thomas Ball and Mooly Sagiv, editors, *POPL*, pages 435–446. ACM, 2011.

[40] Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty session types meet communicating automata. In Helmut Seidl, editor, *ESOP*, volume 7211 of *Lecture Notes in Computer Science*, pages 194–213. Springer, 2012.

[41] Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty compatibility in communicating automata: Characterisation and synthesis of global session types. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *ICALP (2)*, volume 7966 of *Lecture Notes in Computer Science*, pages 174–186. Springer, 2013.

[42] Paul Gastin and François Laroussinie, editors. *CONCUR 2010 - Concurrency Theory, 21th International Conference, CONCUR 2010, Paris, France, August 31-September 3, 2010. Proceedings*, volume 6269 of *Lecture Notes in Computer Science*. Springer, 2010.

[43] Kohei Honda, Vasco Thudichum Vasconcelos, and Makoto Kubo. Language primitives and type discipline for structured communication-based programming. In Chris Hankin, editor, *ESOP*, volume 1381 of *Lecture Notes in Computer Science*, pages 122–138. Springer, 1998.

[44] 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.

[45] Raymond Hu, Nobuko Yoshida, and Kohei Honda. Session-based distributed programming in java. In Jan Vitek, editor, *ECOOP*, volume 5142 of *Lecture Notes in Computer Science*, pages 516–541. Springer, 2008.

[46] ITU-T. Message sequence chart (MSC). Technical Report ITU-T Recommendation Z.120, International Telecommunication Union, 2011.

[47] Fabrizio Montesi Ivan Lanese and Gianluigi Zavattaro. Amending choreographies. In Editors Antonio Ravara, Josep Silva, editor, *Proceedings of WWV 2013, 9th International Workshop on Automated Specification and Verification of Web Systems*, EPTCS, page 15 pages, 2013. to appear.

[48] Walter Knodel. New gossips and telephones. *Discrete Mathematics*, 13(1):95 –, 1975.

[49] Maciej Koutny and Irek Ulidowski, editors. *CONCUR 2012 - Concurrency Theory - 23rd International Conference, CONCUR 2012, Newcastle upon Tyne, UK, September 4-7, 2012. Proceedings*, volume 7454 of *Lecture Notes in Computer Science*. Springer, 2012.

[50] Ivan Lanese, Claudio Guidi, Fabrizio Montesi, and Gianluigi Zavattaro. Bridging the gap between interaction- and process-oriented choreographies. In Antonio Cerone and Stefan Gruner, editors, *SEFM*, pages 323–332. IEEE Computer Society, 2008.

[51] Julien Lange and Alceste Scalas. Choreography synthesis as contract agreement. `http://www.cs.le.ac.uk/people/jlange/papers/agreement.pdf`, March 2013. To appear in ICE'13.

[52] Julien Lange and Emilio Tuosto. A modular toolkit for distributed interactions. In Kohei Honda and Alan Mycroft, editors, *PLACES*, volume 69 of *EPTCS*, pages 92–110, 2010.

[53] Julien Lange and Emilio Tuosto. Synthesising Choreographies from Local Session Types. In Koutny and Ulidowski [49], pages 225–239.

[54] Sjouke Mauw. The formalization of message sequence charts. *Computer Networks and ISDN Systems*, 28(12):1643–1657, 1996.

[55] Bertrand Meyer. Applying ”design by contract”. *IEEE Computer*, 25(10):40–51, 1992.

[56] Robin Milner. *Communicating and mobile systems - the Pi-calculus*. Cambridge University Press, 1999.

[57] Dimitris Mostrous, Nobuko Yoshida, and Kohei Honda. Global principal typing in partially commutative asynchronous sessions. In Castagna [27], pages 316–332.

[58] Matthias Neubauer and Peter Thiemann. An implementation of session types. In Bharat Jayaraman, editor, *PADL*, volume 3057 of *Lecture Notes in Computer Science*, pages 56–70. Springer, 2004.

[59] Nicholas Ng, Nobuko Yoshida, and Kohei Honda. Multiparty session c: Safe parallel programming with message optimisation. In Carlo A. Furia and Sebastian Nanz, editors, *TOOLS (50)*, volume 7304 of *Lecture Notes in Computer Science*, pages 202–218. Springer, 2012.

[60] Nicholas Ng, Nobuko Yoshida, Olivier Pernet, Raymond Hu, and Yiannos Kryftis. Safe parallel programming with session java. In Wolfgang De Meuter and Gruia-Catalin Roman, editors, *COORDINATION*, volume 6721 of *Lecture Notes in Computer Science*, pages 110–126. Springer, 2011.

[61] OASIS. Web services business process execution language (WS-BPEL) version 2.0. `https://www.oasis-open.org/committees/wsbpel`, April 2007.

[62] OMG. Unified modeling language (UML). `http://www.omg.org/spec/UML/`, August 2011.

[63] Luca Padovani. On projecting processes into session types. *Mathematical Structures in Computer Science*, 22(2):237–289, 2012.

[64] SAVARA and testable architecture. `http://www.jboss.org/savara`.

[65] Scribble. `http://www.jboss.org/scribble`.

[66] Alexandra Silva, Simon Bliudze, Roberto Bruni, and Marco Carbone, editors. *Proceedings Fourth Interaction and Concurrency Experience*, volume 59 of *EPTCS*, 2011.

[67] Singularity. `http://research.microsoft.com/projects/singularity/`.

[68] Kaku Takeuchi, Kohei Honda, and Makoto Kubo. An interaction-based language and its typing system. In Constantine Halatsis, Dimitris G. Maritsas, George Philokyprou, and Sergios Theodoridis, editors, *PARLE*, volume 817 of *Lecture Notes in Computer Science*, pages 398–413. Springer, 1994.

[69] Hugo Torres Vieira, Luís Caires, and João Costa Seco. The conversation calculus: A model of service-oriented computation. In Sophia Drossopoulou, editor, *ESOP*, volume 4960 of *Lecture Notes in Computer Science*, pages 269–283. Springer, 2008.

[70] Philip Wadler. Propositions as sessions. In Peter Thiemann and Robby Bruce Findler, editors, *ICFP*, pages 273–286. ACM, 2012.

[71] Yin Wang, Ahmed Nazeem, and Ram Swaminathan. Finding the optimal representation for service composition using the theory of regions. Technical Report HPL-2010-191, HP Laboratories, 2011.

[72] Yin Wang, Ahmed Nazeem, and Ram Swaminathan. On the optimal petri net representation for service composition. In *ICWS*, pages 235–242. IEEE Computer Society, 2011.

[73] Nobuko Yoshida, Pierre-Malo Deniélou, Andi Bejleri, and Raymond Hu. Parameterised multiparty session types. In C.-H. Luke Ong, editor, *FOSSACS*, volume 6014 of *Lecture Notes in Computer Science*, pages 128–145. Springer, 2010.