Interface Automata for Choreographies

Hao Zeng

School of Informatics
University of Leicester

A thesis is submitted for the degree of

Doctor of Philosophy

October 2019
Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 39,000 words including bibliography, footnotes, tables and equations and has fewer than 60 figures.

Hao Zeng

October 2019
Acknowledgements

First of all, I would like to give my immeasurable appreciation to my supervisor Dr. Emilio Tuosto, who introduced me to the area of choreography, for his very patient guidance, excellent knowledge, valuable suggestion, constant encouragement and modest, open-minded personality, he inspire me both in academic research and daily life.

Besides Emilio, I would like to express my sincere gratitude to Prof. Alexander Kurz, he had guided me for three years with lots of good ideas and advice. I learned how to think mathematically from him. Also, I would like to express my heartfelt thanks to Dr. Rumyana Neykova and Prof. Mohammad Mousavi for insightful discussions, advice and comments.

After that, I would like to give my thanks to all the staff in the department who have helped me, who have cooperated with me during these years. I am very glad to be a graduate teaching assistant working with all of you. The enjoyable experiences of teaching are the great values for my future career and academic research.

Then, I would like to dedicate this thesis to my loving parents who support me for nearly six years oversea study from M.Sc to Ph.D in United Kingdom. They cultivate me from a simple, naive child to a better person who is able to explore the ocean of science and nature.

Last but not least, I would like to express my special thanks to my friends in Leicester. Because of you, I have spent six wonderful years in Leicester.
Abstract

Formal choreographic models yield a suitable frameworks to specify, analyse, and develop message-passing applications. In this application domain, orchestration has so far been preferred to choreography because the former offers a simpler approach to tame the complexity of developing distributed applications. On the one hand, orchestration yields a toolbox of handy and relatively simple solutions to practitioners. However, orchestration shows scalability limitations that choreographies seem able to tackle. Another key advantage of choreographies is the possibility of guaranteeing behavioural properties of applications “by construction”. Correct components can be (semi)automatically derived from correct global specifications. In this context it is crucial to verify the correctness of global specifications.

In this thesis, we adopt the global choreographies of Guanciale and Tuosto to specify global views while we abstract components as special automata. More precisely, we generalise the class of interface automata of de Alfaro and Henzinger to group interface automata and equip them with two internal products, instrumental to the verification of “sound” global specifications. In particular, these products allow us to characterise erroneous states that potentially spoil interactions. A main result of our approach is that the absence of those erroneous state is equivalent to well-formedness of global choreographies, that is a sufficient property guaranteeing well-behaviour of local components projected from the global specification.
# Table of contents

List of figures vii

1 Introduction 1

1.1 Global and local view of choreographies 2
1.2 Main contributions 7
1.3 Structure of the Thesis 8

2 Related Work 10

3 Background 15

3.1 A syntax of global views 15
3.2 Semantics of global views 19
3.3 Well-formedness of g-choreographies 24
    3.3.1 Well-sequencedness 26
    3.3.2 Well-forkedness 27
    3.3.3 Well-branchedness 28
    3.3.4 Another well-branchedness 33
3.4 Interface automata 39

4 Generalising Interface Automata 52

4.1 Motivations 52
4.2 Group interface automata 53
4.3 Composing GIAs 55
# 5 Verifying G-Choreographies with GIA

5.1 Another semantics of g-choreographies ........................................ 72
5.2 GIA-based Verification: Theory .................................................. 74
  5.2.1 Projections of GIA ......................................................... 75
  5.2.2 Removability of $\tau$-transitions ......................................... 80
  5.2.3 Branching error state ........................................................ 87
  5.2.4 Parallel error states ......................................................... 92
5.3 Auxiliary lemmas ........................................................................ 92
5.4 GIA-based Well-formedness ....................................................... 100
5.5 A Few Examples ....................................................................... 110
  5.5.1 Examples of sequential composition ....................................... 110
  5.5.2 Examples of parallel composition .......................................... 113
  5.5.3 Examples of branching composition ...................................... 116
  5.5.4 Examples of iteration .......................................................... 122

# 6 A Prototype for GIA

6.1 An Implementation of GIA .......................................................... 125
6.2 Applying GIAGG ....................................................................... 127

# 7 Conclusion and Future Work

References
List of figures

1.1 Orchestration .................................................. 2
1.2 Choreography .................................................. 3
1.3 A global view of an on-line shopping choreography .......... 3
1.4 Two variants of the on-line shopping application as g-choreographies . 5
1.5 An interface automata for the Buyer $B$ ........................ 7

2.1 A variant of well-sequencedness .............................. 11
2.2 A branching composition ..................................... 11

3.1 The visual notations of g-choreographies .................. 16
3.2 Instances of sequential, parallel, branching and iteration ...... 18
3.3 Instances of incorrect branching compositions .............. 19
3.4 Typical instances of sequential compositions ............... 22
3.5 Instances of pomsets for branching and parallel compositions 23
3.6 The pomsets for the online shopping global choreography .... 24
3.7 Instances of well-sequencedness .............................. 27
3.8 Instances of well-forkedness .................................. 28
3.9 Instances of well-branchedness ............................... 30
3.10 A general well-branched branching composition ............ 33
3.11 An interface automata of Buyer .............................. 40
3.12 The product of three composable interface automata .......... 43
3.13 Instance of error states in the product ............... 44
3.14 Instances of removing incompatible states and error states 46
List of figures

4.1 Error states in the product ........................................ 53
4.2 An instance of GIA .................................................. 55
4.3 $\otimes$-product among three composable GIA .................. 59
4.4 Deadlock states .................................................... 60
4.5 Unspecified reception states ..................................... 61
4.6 Error states in $\otimes$-product of GIA ....................... 63

5.1 An example of pomsets of a simple iteration .................... 73
5.2 Instances of a variant of well-sequencedness ................... 73
5.3 Research approach on naive well-formedness .................... 75
5.4 Instances of projection from g-choreographies .................. 79
5.5 Spurious error states .............................................. 80
5.6 Removability of $\tau$-transitions in projections ............... 82
5.7 Removability of $\tau$-transitions in projections (cont.) ..... 83
5.8 Refined GIA and their $\otimes$-product .......................... 84
5.9 Branching bisimulation between GIA ............................ 86
5.10 Weak bisimulation between GIA ................................. 87
5.11 Verification of the well-formedness of $G_1$ ................. 88
5.12 Verification of the well-formedness of $G_2$ ................... 88
5.13 Verification of the well-formedness of $G_3$ ................... 89
5.14 Verification of the well-formedness of $G_4$ ................... 90
5.15 Parallel error state ............................................... 93
5.16 A well-formed sequential composition $G_1$ .................... 111
5.17 Verification of the well-formedness of $G_1$ .................... 112
5.18 A well-formed sequential composition $G_2$ .................... 112
5.19 Verification of the well-formedness of $G_2$ .................... 113
5.20 A well-formed parallel composition $G_3$ .................... 114
5.21 Verification of the well-formedness of $G_3$ .................... 115
5.22 A not well-formed parallel composition $G_4$ ............... 115
5.23 Verification of the well-formedness of $G_4$ .................... 116
5.24 A well-formed branching composition $G_5$ .......................... 117
5.25 Verification of the well-formedness of $G_5$ .......................... 118
5.26 A not well-formed branching composition $G_6$ .......................... 118
5.27 Verification of the well-formedness of $G_6$ .......................... 119
5.28 A not well-formed branching composition $G_7$ .......................... 120
5.29 Verification of the well-formedness of $G_7$ .......................... 121
5.30 A well-formed iteration $G_8$ .......................... 123
5.31 Verification of the well-formedness of $G_8$ .......................... 124

6.1 Two g-choreographies $G$ and $G'$ .......................... 128
6.2 GIA projected from $G$ and $G'$ .......................... 129
6.3 Refined GIA without removable $\tau$ .......................... 129

7.1 A naive well-branchedness instance .......................... 133
7.2 Another way of projections .......................... 133
7.3 A not general well-branchedness instance .......................... 134
7.4 Research conjecture on repairing g-choreographies .......................... 135
Chapter 1

Introduction

Nowadays distributed applications are widely used in our daily life, ranging from mobile payments to social communications to web services to multi-core computing. Also, distributed applications gradually become a trend to lead the direction of software development. Therefore, communications among each component of distributed applications are the key elements of developments. Formalising the coordination of distributed components becomes an important challenge. The use of abstract models to tame the complexity of distributed applications starts to be adopted also in industrial context [8]. And this trend is also leading to new approaches to software development [20].

To support the development of distributed applications, the two main methodologies are orchestration and choreography.

Orchestration [47] prescribes that a centralized executable orchestrator coordinates the interactions among different participants. The orchestrator plays an important role to invoke and combine all the participants by a fixed logic. In other words, the relationship between all the participants are represented by the orchestrator. The orchestrator tells what, when, and which actions each component should perform. For instance, Fig. 1.1 gives an intuition of how orchestration works. The orchestrator invokes participants and participants reply the orchestrator to participate in the compositions. Substantial effort has been dedicated to properly compose [2, 44, 39], to adapt [40] participants to orchestrations, and check the correctness of implementation
of orchestrations [9]. As a matter of fact, orchestration is a bottom-up approach and local level (single party perspective) of composition the activities of participants. An important support to orchestration is provided by the business process execution language (BPEL) [1]. A BPEL orchestrator realises the logic which invokes several participants and combines their responses according to the application at hand. A typical structure of a BPEL system is given in Fig. 1.1.

Choreography [33] employs a decentralized approach for the composition of distributed components; this approach does not require a centralized orchestrator. Roughly speaking, choreographies specify the message exchanges or communication contracts among the participants in distributed systems. For instance, Fig. 1.2 describes an intuition of how choreography works. Participants interact by send or receive messages among each other.

In this thesis, we focus on choreographic models, which we now describe in more detail.

1.1 Global and local view of choreographies

A choreography provides two views of a distributed application, the so-called global and local views [33]. The global view describes the application from a holistic level where interactions form a comprehensive body for the coordination of components.
Global views abstract away from the underlying communication details and have interactions as units of coordination. The benefit is that we are able to observe the whole communication processes rather than focusing on the behaviours of one single communication participant. Additionally, interactions are oblivious to the asynchrony in the communication: in the global view interactions are atomic primitives. This makes it simpler to reason about the communication logic of the application.

There are many languages to describe global views. For the sake of the presentation we consider an example describing an on-line shopping application using message sequence charts [30] (MSC for short), a model proposed by International Telecommunication Union in 1994. Fig. 1.3 gives two MSCs describing two scenarios of our on-line shopping application. The left-most MSC consists of three participants named Buyer, Seller and Shipper; five interactions described by arrows between participants. The diagram imposes an order on the interactions; for instance, the request from Buyer to Seller, precedes the offer from Seller to Buyer and, likewise, the interaction pay from

![Fig. 1.2 Choreography](image)

![Fig. 1.3 A global view of an on-line shopping choreography](image)
1.1 Global and local view of choreographies

Buyer to Seller precedes the interaction deliveryInfo from Seller to Shipper and so on. Note that this MSC describes also an implicit flow of information through the system. For example, the Seller may receive in the interaction request information about the goods the Buyer is interested in. This allows the Seller to formulate on offer to the Buyer, and so on until a delivery is made for the Buyer.

The right-most MSC in Fig. 1.3 models an alternative protocol where the Seller cannot make an offer because the requested goods are not available. In this case, the interaction notAvailable should be fired. Then, the protocol continues with the interaction noInfo after which it terminates. Notice that these MSCs highlight the communication pattern while hiding local computation (for instance, the right-most MSC could be executed because the goods are not in the stock or because they have been discontinued).

Models like MSCs are not ideal for our purposes. The main reasons is that it is not easy to automatically attain local views from them. We therefore adopt global choreographies (g-choreographies for short) [24, 25], a formalisation of global views for message-passing systems amenable of begin projected on local views. Also, g-choreographies are rather expressive (see the discussion in [24, 25]) and have an abstract semantics based on pomsets [27, 26]. Fig. 1.4(a) shows a g-choreography representing the same on-line shopping application global described by the MSC in Fig. 1.3. Note that the g-choreography of this simple protocol clearly specifies where the choice made by S takes place and where it finishes. Also, observe that the choice is locally made by S and propagated to the other participants.

A well-formed g-choreography is deadlock free [24, 25] (cf. Chapter 3.3). We give an intuition account of well-formedness. In particular, we focus on what this condition imposes on distributed choices. The key point is that a distributed choice requires the coordination of all components in order for each of them to follow the same branch of the choice. We illustrate the problem through the examples in Fig. 1.4(a) and Fig. 1.4(b), where we use a graphical notation for g-choreographies that we clarify in Chapter 3. This notation is basically composing interactions (eg. $\mathsf{B} \xrightarrow{\text{request}} S$)
sequentially of using gates to represent distributed choices (the diamond nodes in the graphs) or parallel composition of communication protocols (not represented in the figures).

(a) A well-branched g-choreography  (b) A non well-branched g-choreography

Fig. 1.4 Two variants of the on-line shopping application as g-choreographies

The difference between the g-choreography in Fig. 1.4(a) and the one in Fig. 1.4(b) is that the latter lacks the interaction $S \rightarrow H$ in the right branch. As we discuss below this is crucial for compromising the correctness of the g-choreography in Fig. 1.4(b).

The g-choreography in Fig. 1.4(a) is well-formed since the communication pattern allows each participant to unequivocally identify which branch of the choice is taken. In fact, participant $S$ (non-deterministically) decides which branch to take and communicates its choice to $B$ by sending it different messages. Eventually, also participant $H$ receives different messages on each branch so that it can discriminate which choice was taken by $S$. Following this communication pattern, all the interactions are aligned and it is not possible that some components to take one branch while others follow the other branch.

Let us now consider the g-choreography in Fig. 1.4(b); as anticipated, this is g-choreography is not well-formed. In fact, assume that $B$ sends message request to $S$, 
then $S$ decides to reply \texttt{notAvailable} to $B$, which is the selection of the right branch in Fig. 1.4(b). After that, $B$ and $S$ terminate. However, $H$ has no idea about what happens since $B$ and $S$ did not send any messages to $H$, therefore, $H$ will wait for the interaction from $S$ forever and will not terminate. This causes $H$ to deadlock waiting for the \texttt{deliveryInfo} message that is not going to be sent.

Now we turn our attention to the local view of choreographies. Local views of choreographies describe the local behaviours of each participant involved in terms of the send/receive actions they perform.

As said, given a g-choreography, the local view corresponding to each participant can be obtained by projection. This can be represented by the following diagram:

\begin{equation}
\text{Global View} \xrightarrow{\text{projection}} \text{Local View}
\end{equation}

Interface automata [13, 15] are used to formalise interaction mechanisms in component-based design. Fig. 1.5 shows an interface automaton $B$ which represents the local behaviours of Buyer in the g-choreography in Fig. 1.4(a). Basically, an interface automaton consists of a finite state automaton equipped with an interface capturing the interactions a component can “provide” to its environment and those the component is capable to “consume” from the environment. In the example of Fig. 1.5, the rectangle specifies the scope of the interface automaton and the down and up arrows on the edge of the rectangle describe the output and input interfaces respectively (for more details see Chapter 3.4). In Fig. 1.5, the transition actions \texttt{request!} and \texttt{offer?} represent the interface automaton $B$ sending message \texttt{request} to environments (also interface automata), and receiving \texttt{offer} from environments. However, we consider that interface automata is not an appropriate local model for g-choreography since they are synchronous (for more details see Chapter 4) and there is less information about the participants. For instance, in Fig. 1.5, we just know $B$ intends to send message \texttt{request} to other, but we do not know who should receive \texttt{request}. However, the sender and receiver of messages should be certain in g-choreographies.
1.2 Main contributions

Therefore, we generalise a class of interface automata [13, 15] and show how g-choreographies can be projected to represent local views as sets of such automata. Moreover, we demonstrate that local views can be used to check the correctness of g-choreographies. More precisely, we show how our interface automata can be used to check the well-formedness of g-choreographies.\footnote{As known from the literature, well-formedness of g-choreographies guarantees well-behaviour of local components projected from the global specification (see e.g. [29, 36, 24, 25]).} Technically, we define two internal products to the class of our interface automata and show that checking some conditions on these products is equivalent to checking for the well-formedness of g-choreographies.

We generalise interface automata [13, 15] by defining group interface automata. The group interface automaton $\mathcal{I}_B$ for the local view of buyer $B$ of Fig. 1.4(a) is

Roughly speaking, similar to interface automata, the rectangle also specifies the scope of the machine, the down and up arrows on the edge of the rectangle describe the output and input interfaces respectively (for more details see Chapter 4).

1.2 Main contributions

This thesis is based on the choreographic framework presented in [24, 25]. We elaborate on the pomset semantics for global specifications (g-choreographies) given in [24, 25]
and formalise the interplay between global and local specifications. More precisely, we reduce the notion of well-formedness of g-choreographies given by Guanciale and Tuosto to the analysis of (an extension of) interface automata [13]. The notion of well-formedness identifies a sufficient condition to guarantee that the asynchronous execution of the projections of a g-choreographies is sound, that is the execution is deadlock-free and without orphan messages (meaning that no message stays in the communication channels after all participants terminate) nor unspecified receptions [10].

The main contribution of this thesis is the reduction of well-formedness of g-choreographies the absence of error-states in a variant of interface automata. More precisely we show that a g-choreography $G$ is well-formed if, and only if, the group interface automaton consisting of $G$’s projections does not contain error-states. This result requires some technical contributions.

Firstly, we generalise interface automata to group interface automata (GIA) to represent local views of global choreographies. Secondly, we define a product operation on GIA that allows us to identify a class of configurations, dubbed error states that may spoil communications. The identification of error-states is based on the notion of removable internal transition of GIA that we define here.

1.3 Structure of the Thesis

The material in this thesis is organised as follows.

Chapter 2 reviews the related work of choreography models.

Chapter 3 lists the background knowledge on both syntax and semantics of g-choreographies, the well-formedness of g-choreographies, and interface automata.

Chapter 4 introduces a generalisation of interface automata called group interface automata.

Chapter 5 introduces the group interface automata based verification on a variant of g-choreographies.

Chapter 6 introduces a prototype of implementation for group interface automata.
Chapter 7 summarises the research in the thesis, gives an intuition about group interface automata based verification applying a more general semantics of g-choreographies and suggests an idea of automatic repairing of not well-formed g-choreographies to well-formed versions.
Chapter 2

Related Work

Global choreography [24, 25] has been proposed as an expressive global model to describe the global view of choreographies. A general notion of well-formedness has also been defined in [24, 25] as well. This notion relies on the formal abstract semantics given in [24, 25] and it has been related to communicating finite state machines [10], used there to represent the local views of g-choreographies. More precisely, [24, 25] give a theorem showing that well-formed g-choreographies are deadlock free and can be correctly realised by systems of communicating finite state machines obtained by projection from the g-choreographies.

The semantics of g-choreographies is more abstract since it is not based on traces and it makes minimal assumptions on message exchange at lower levels (see Section 3.2). According to [24, 25], the semantics of global choreographies provide a more expressive framework since allow participants to appear in both threads of parallel compositions (see Section 3.3.2) and passive participants do not need to receive a message signalling the selected choice as first operation in branching compositions (see Section 3.3.4).

In this thesis, for simplicity, we use a variant of g-choreographies and its pomset semantics as the global view of choreographies. The variant disregards the requirements of well-sequencedness, and borrows the well-forkedness, the well-branchedness imposed in [24, 25]. For instance, the variant uses a relaxed notion of well-branchedness for simplicity.
A sequential composition $G$ with its pomset semantics in our variant are shown in Fig. 2.1. According to [24, 25], the semantics of $G$ is undefined since $G$ violates well-sequencedness due to the lack of causal dependencies among the participants of the two interactions. Instead, our variant gives $G$ the semantics represented by the pomset shown in the right-hand-side of the figure, which simply allows $A \rightarrow B$ and $C \rightarrow D$ to run concurrently. Therefore, the semantics of $G$ is equivalent to the semantics of the parallel composition $A \rightarrow B | C \rightarrow D$.

Then, we briefly discuss two classes of well-branchedness based on the syntax and semantics respectively. For instance, Fig. 2.2 shows a branching composition $G = A \rightarrow B; B \rightarrow A + A \rightarrow B; B \rightarrow A$. According to the syntax of $G$, participant $A$ does the same behaviours sending $m$ to $B$ in both branches at initial, therefore, $G$ is not well-branched (see the detail on Section 3.3.3). And from the local point of views, there exists a potential problem in the interaction between $A$ and $B$ since the message
m sent by left branch \(G'\) may be “stolen” by the right branch \(G''\), then participant \(B\) will send message \(y\) to \(A\), while \(A\) expects to receive \(x\). This makes \(y\) an orphan message and participant \(A\) waiting for message \(x\) forever.

However, according to the semantics of \(G\), \(G\) is general well-branched (see the detail on Section 3.3.4). Participant \(A\) does the same send actions in both branches, then \(A\) receives messages \(x, y\) from \(B\) in \(G'\) and \(G''\) respectively. Dually, participant \(B\) initially receives same messages \(m\) in both branches, then starts to send messages \(x, y\) to \(A\) in \(G'\) and \(G''\) respectively. Here, we factor out the same behaviour in both branches at the branching start point since they are common branches. Therefore, the point of \(A\) (resp. \(B\)) starting to receive (resp. send) \(x, y\) from (resp. to) \(B\) (resp. \(A\)) is the real divergence point. Then, from local point of views, there is no problems in the interaction between \(A\) and \(B\).

As a summary, in general, given a branching composition \(G\), if we adopt well-branchedness (Definition 3.3.4), it is easy to obtain the well-formedness of \(G\) since we do not need to consider the common prefix. And at the local level, we do not need to determinise the local models of each participant (eg. CFSMs) when do the verification. However, if we use general well-branchedness (Definition 3.3.9), the complexity of checking the well-formedness of \(G\) increases since to compute the common prefix takes times, and the local models have to be determinised before doing the verification. In addition, to obtain the determinised local models (eg. CFSMs) has an exponential theoretical complexity. This is mainly due to the minimisation of finite state machines.

Next, we review the choreographic models that are closest to the model presented in this thesis.

*Global type* or *global session type*, is proposed firstly in [28], then refined in [29]. Global types use algebraic expressions to explicitly describe the global choreographic specification. By projection, global types are able to obtain statically type-checked local types which are the local specification of participants. Well-typed guarantees the correct interactions such as type safety, deadlock freedom, and progress in global types. Global type is also applicable to protocol programming contexts [12, 50, 48, 45].
The semantics of global types can be defined under a class of approaches, where the semantic of global view is constructed by composing the semantics of its corresponding local views. Process algebras equipped with an operational semantics are existing between global type and its corresponding local types.

*Global graph* or *graph-based global type*, is firstly proposed in [18] as a graphic extension of generalising the syntax of previous systems [4, 28]. Compared with global type, global graph explicitly distinguishes the branching points from the forking points in order to avoid the state explosions. Global graph is also refined in [24, 25] as global choreographies.

*Conversation protocol automaton* [22, 3] is a kind of models of finite state machines adapted in the research of deciding the realizability of choreographies, but the realizability of choreographies has been proved as undecidable by [19] since the synchronizability [23, 49, 34] can not guarantee the realizability of choreographies. As a matter of fact, a conversation protocol automaton is not a model of true concurrency. Instead, interleaving is used to simulate parallel compositions.

*Message sequence chart (MSC)* [30] is a graphical model standardized by International Telecommunication Union in 1994. MSC provides intuitive and transparent message exchanges between the communication system components. So MSC is also an effective methodology for supporting the specification and the implementation for communication systems. The formalisation for the semantics of MSC have been studied in [17, 41], then a process algebra based semantics is accepted for standardization [42, 31]. The algebraic semantics are easy to express the meaning of MSC.

Next, we review the local models. *Interface automata* [13, 15] are a class of synchronous local models to support component-based design and verification in software engineering. Two composable interface automata are allowed to interact via a product that respects their interfaces. An error state \((v, u)\) in a product automaton consists of a state \(v\) in which a send in the shared output interface will not be consumed immediately from the corresponding state \(u\) onwards. However, the notion of error states in interface automata is too strong for interface automata as local views of global
choreographies. *Communicating finite state machines (CFSM)* [10] are a convenient setting to analyse choreographies from a local point of view. However, CFSM do not have interfaces and product operations. In order to combine advantages from interface automata and CFSM, we propose *group interface automata* as an extension to interface automata. Group interface automata strengthen the sender and receiver on interfaces, add notions of special internal $\tau$-transitions, and redefine products and error states. These changes allow us to adopt interface automata to analyse global choreographies.
Chapter 3

Background

This chapter reviews the background concepts forming the basis of this thesis. We start with the syntax of g-choreographies, then we recall their pomsets semantics. Finally, we review interface automata that we adopt as our model of local views.

3.1 A syntax of global views

As stated in Chapter 1, global and local views of choreographies are the interpretation of choreographies from two different perspectives. Now, let us review a syntax of global views called global choreography [24, 51]. We simplify the original definition since we will not use control points. While we add the iterations to fulfil the structure of the definition. The semantics of global views can be suitably given in terms of partial orders of communication events. In particular, we follow the approach of Guanciale and Tuosto who gave a pomset semantics of g-choreographies in [24, 51]. Let \( \mathcal{P} \) be a set of participants, \( \mathcal{M} \) be a set of messages.

**Definition 3.1.1** (Global Choreography [24, 51]). A global choreography (g-choreography for short) is a term \( G \) derived by the following grammar:

\[
G ::= 0 \mid A \xrightarrow{m} B \mid G; G' \mid G|G' \mid G+G' \mid \{G\} @A.
\]
Where the empty global choreography is $0$; $A \xrightarrow{m} B$ is an interaction whereby message $m \in \mathcal{M}$ is sent from participant $A \in \mathcal{P}$ to participant $B \in \mathcal{P}$ ($A \neq B$); the operators $\_;\_$, $\_|\_$, and $\_+\_$ allow us to compose g-choreographies sequentially, in parallel and in non-deterministic branches respectively; the $\ast\{\_\}@$ encloses the global graph which repeats executions (iterations) for an unspecified number of times; here the second argument $A \in \mathcal{P}$ (called decider) is the participant who decides the termination of the iteration. As a matter of fact, in order to inform other participants to continue or terminate the iteration, the decider will send special messages ($lp_i$, $ex_i$ corresponding to continuation, termination) to each rest participants. However, such special interactions are not represent in iterations of g-choreographies. Instead, these sending/receiving actions will be represented in the local views of choreographies, see Chapter 5 for detail.

In addition, each g-choreography has a specific corresponding visual notation called global graph which is depicted in the following Fig. 3.1.

Next, we elaborate the details of the visual notations shown above. The circled and the double circled nodes in each graph represent the unique initial and terminal node respectively.
(1) The empty g-choreography 0 includes two nodes which are the initial node and the terminal node. While the arrow connecting these two nodes means the global graph starts from the initial node, then ends at the terminal node.

(2) The interaction \( A \xrightarrow{m} B \) consists of three nodes which are the initial node, the boxed node representing the basic communication unit \( A \xrightarrow{m} B \) called the interaction, and the terminal node; two arrows which show the g-choreography \( A \xrightarrow{m} B \) starts from the initial node to the interaction node of \( A \xrightarrow{m} B \), then end at the terminal node.

(3) The sequential composition \( G; G' \) consists of the graphs of \( G \) and \( G' \) where the terminal node of \( G \) and the initial node of \( G' \) are merged.

(4) The parallel composition \( G \parallel G' \) consists of six nodes which are the initial node, two small boxed nodes representing the beginning of parallel and the end of parallel respectively, the graphs of \( G \) and \( G' \). The graphs \( G \) and \( G' \) are able to concurrently execute as two threads under the parallel composition \( G \parallel G' \).

(5) The branching composition \( G + G' \) consists of six nodes which are the initial node, two small diamond nodes representing the beginning of branching and the end of branching respectively, the graphs of \( G \) and \( G' \). Only one of graphs \( G \) and \( G' \) is able to execute under the branching composition \( G + G' \).

(6) The iteration \( \ast\{G\}@A \) consists of five nodes which are the initial node, two small diamond nodes enclosing the body of loops, the boxed node representing the body of the iteration, and the terminal node; five arrows which represent the g-choreography \( \ast\{G\}@A \) starts from the initial node to the beginning of the loop, then the g-choreography \( \ast\{G\}@A \) executes the iteration body of \( G \) till the end of execution of \( G \), after that, the participant \( A \) decides the g-choreography \( \ast\{G\}@A \) continues to execute \( G \) or leaves the iteration to terminate, where participant \( A \) is not represented.

We demonstrate four simple instances to show a sequential composition, a parallel composition, a branching composition and an iteration respectively in Fig. 3.2.

(1) The sequential composition \( G_1 = A \xrightarrow{m} B; B \xrightarrow{m} C \) has four participants \( A, B, C, D \) and two messages \( m, n \) involved. \( G_1 \) specifies that the message \( m \) sent from the
3.1 A syntax of global views

participant $A$ to the participant $B$ has to take place before the message $n$ sent from the participant $B$ to the participant $C$, then terminates.

(2) The parallel composition $G_2 = A \xrightarrow{m} B \mid B \xrightarrow{n} C$ has three participants $A$, $B$, $C$ and two messages $m$, $n$ involved. $G_2$ specifies the message $m$ sent from the participant $A$ to the participant $B$ goes simultaneously with the message $n$ sent from the participant $B$ to the participant $C$, after that, the global graph terminates.

(3) The branching composition $G_3 = A \xrightarrow{m} B + A \xrightarrow{n} B$ consists of two participants $A$, $B$ and two messages $m$, $n$. $G_3$ specifies either the message $m$ sent from the participant $A$ to the participant $B$ or the message $n$ sent from the participant $A$ to the participant $B$, then, terminates.

(4) The iteration $G_4 = \ast \{A \xrightarrow{m} B\} @ A$ consists of participants $A$ and $B$ repeatedly exchanging message $m$. Participant $A$ decides when to halt iteration $G_4$. More precisely, after participant $A$ sends message $m$ to $B$, participant $A$ will send a special message (continuation or end) to notify $B$ the decision of $A$. If $B$ receives the message identifying to continue, then $B$ goes to the initial state waiting for message $m$ from $A$ again. If $B$ receives the message identifying to end, then $B$ terminates.

Let us review the branching composition $G_3 = A \xrightarrow{m} B + A \xrightarrow{n} B$ in Fig. 3.2. Here, both participants $A$ and $B$ eventually terminate because they realise which branch to execute from the messages exchanges. Therefore, informally, we call that $G_3$ is a correct branching composition.

Conversely, Fig. 3.3 shows two instances of incorrect branching compositions.
3.2 Semantics of global views

Guanciale and Tuosto proposed a semantics of g-choreographies in terms of pomsets [24, 51] that we now review.

Pomsets for g-choreographies are based on send actions and receive actions on channels. Formally, the set of channels is defined as \( C = \{ (A, B) \mid A, B \in \mathcal{P}, A \neq B \} \),

Fig. 3.3 Instances of incorrect branching compositions

(1) In \( G_5 \), participant \( A \) decides to communicate either with \( B \) or with \( C \) messages \( m \) and \( n \) respectively. Whatever \( A \) decides to do, one of the other participants cannot ascertain what to do. In fact, if \( A \) sends e.g., message \( m \) to \( B \) then \( C \) may decide to keep waiting for message \( n \) (since \( C \) is not informed about the interaction between \( A \) and \( B \)).

(2) Also \( G_6 \) cannot be realised distributively; the reason is that participant \( A \) and participant \( C \) are independent and uncoordinated and therefore none of them is aware of the decision taken by the other. Therefore in a distributed implementation of \( G_6 \) both \( A \) and \( C \) may not send any message (because they assume that the other is sending) or they may both decide to send their message to the partners (because they assume that other is not sending).

From the two instances shown in Fig. 3.3, roughly speaking, the decision about the choice is not correctly propagated, possibly leading some participants to execute a wrong branch.

### 3.2 Semantics of global views

Guanciale and Tuosto proposed a semantics of g-choreographies in terms of pomsets [24, 51] that we now review.
the channel $AB$ is short for the $(A, B) \in C$. The set of labels $\mathcal{L}$ is defined by
\[
\mathcal{L} = \mathcal{L}^1 \cup \mathcal{L}^2 \quad \text{where} \quad \mathcal{L}^1 = \mathcal{C} \times \{!\} \times \mathcal{M} \quad \text{and} \quad \mathcal{L}^2 = \mathcal{C} \times \{?\} \times \mathcal{M},
\]

where $\mathcal{L}^1$ is the set of outputs representing send actions, $\mathcal{L}^2$ is the set of inputs representing receive actions. Hereafter, $AB!m$ and $AB?m$ shorten $(AB, !, m) \in \mathcal{L}^1$ and $(AB, ?, m) \in \mathcal{L}^2$ respectively. The subject of an action is defined by
\[
sbj(AB!m) = A \quad \text{and} \quad sbj(AB?m) = B
\]

Recall that a binary relation is a partial order if it is (1) reflexive, (2) anti-symmetric, and (3) transitive.

\textbf{Definition 3.2.1 (Lposet [24, 51]).} A labelled partially ordered set $r$ for global choreographies ($lposet$) is a triple $(E, \leq, \lambda)$, with $E$ a set of events, $\leq \subseteq E \times E$ a partial order on $E$, and $\lambda : E \to \mathcal{L}$ a labelling function.

Note that Definition 3.2.1 permits to have $\lambda(e) = \lambda(e')$ for two $e, e' \in E$ s.t. $e \neq e'$, namely two event occur in different places with the same action. The relation $\leq$ describes partial orders, in other words, $\leq$ represents the causal dependencies among events. We use $e \rightarrow e'$ to denote $e \leq e'$ and let $\varepsilon$ denote the empty lposet.

\textbf{Definition 3.2.2 (Isomorphic class of lposet [24, 51]).} Two lposets $(E, \leq, \lambda)$ and $(E', \leq', \lambda')$ are isomorphic iff there exists a bijection $\phi : E \to E'$ such that $e \leq e'$ iff $\phi(e) \leq' \phi(e')$ and $\lambda = \lambda' \circ \phi$.

\textbf{Definition 3.2.3 (Pomset [24, 51]).} A partially-ordered multi-set $[E, \leq, \lambda]$ (of actions), pomset for short, is an isomorphism class of lposet $(E, \leq, \lambda)$.

The advantage of using lposets to describe g-choreographies is that the partial orders are able to explicitly represent the causal dependencies among communications.
Given a basic interaction $G = A \xrightarrow{m} B$, the pomset for $G$ is written as follows.

$$\llbracket G \rbracket = \{[\mathcal{E}, \leq, \lambda]\}$$

$$= \{[[\{e_1, e_2\}, \{(e_1, e_1), (e_1, e_2), (e_2, e_2)\}, \lambda]]\}$$

where $\lambda = \begin{cases} e_1 \mapsto AB!m \\ e_2 \mapsto AB?m \end{cases}$

We use $\llbracket G \rrbracket$ to denote the set of pomsets for $G$. In fact, the interaction $A \xrightarrow{m} B$ is represented by a set of pomsets, where $[\mathcal{E}, \leq, \lambda]$ is the only element. The pomset $[\mathcal{E}, \leq, \lambda]$ consists of a set of events $\mathcal{E} = \{e_1, e_2\}$, the partial order relation $\leq$ requiring that $e_2$ is casually dependent on $e_1$, and the labelling function $\lambda$ assigning labels $AB!m$ and $AB?m$ to events $e_1$ and $e_2$ respectively. In summary, the pomset for interaction $A \xrightarrow{m} B$ shows the send action $AB!m$ must happen before the receive action $AB?m$.

Pomsets have a visual notation that omits the reflective relations of events and uses labels instead of events. For instance, the pomset for the interaction $G = A \xrightarrow{m} B$ can be represented as follows.

$$\left[ AB!m \rightarrow AB?m \right]$$

We have shown pomsets for interactions. Next, we start to review pomsets for sequential compositions, parallel compositions, branching compositions and iterations. Given a natural number $n$, $\mathbf{n}$ represents the singleton $\{n\}$. Use $X \uplus Y$ to represent the disjoint union of two sets $X$ and $Y$: $X \uplus Y = (X \times 1) \cup (Y \times 2)$, where $\_ \times \_$ are the Cartesian product between two sets. Then, given a function $f$ on $X$, defines $f \otimes \mathbf{n} = \{(x, n) \mapsto f(x) \mid x \in X\}$ as the function extending $f$ to $X \times \mathbf{n}$; analogously, for a relation $R \subseteq X \times Y$, let $R \otimes \mathbf{n} = \{((x, n), (y, n)) \mid (x, y) \in R\}$ be the relation extending $R$ to $(X \times \mathbf{n}) \times (Y \times \mathbf{n})$.

**Definition 3.2.4 (Pomsets for sequential compositions [24, 51]).** Let $r = [\mathcal{E}, \leq, \lambda]$ and $r' = [\mathcal{E}', \leq', \lambda']$ be two pomsets. For a pomset $r$ and a participant $A \in \mathcal{P}$, let $\mathcal{E}_{r, A} = \{e \in \mathcal{E}_r \mid \text{sbj}(\lambda_r(e)) = A\}$ be the set of events of $A$ in $\mathcal{E}_r$. The pomset for the
3.2 Semantics of global views

**sequential composition** \(\text{seq}(r, r')\) of \(r\) and \(r'\) is defined as

\[
\text{seq}(r, r') = [E \sqcup E', \leq_{\text{seq}}, (\lambda \otimes 1) \cup (\lambda' \otimes 2)],
\]

where

\[
\leq_{\text{seq}} = \left( (\leq \otimes 1) \cup (\leq' \otimes 2) \cup \bigcup_{A \in P} ((E_{r,A} \times 1) \times (E_{r',A} \times 2)) \right)^*.
\]

and \(*\) is the reflexive-transitive closure.

Essentially, the pomsets for sequential composition \(\text{seq}(r, r')\) defined above retains the original causal dependencies in both \(r\) and \(r'\) and builds new causal dependencies between the communication events of \(r\) and those of \(r'\) done by the same participant. Fig. 3.4 summarizes the typical causal dependencies (denoted by dashed arrows) in sequential compositions.

\[
\begin{align*}
&\text{G}_a = A \xrightarrow{m} B; A \xrightarrow{n} C, \\
&\text{G}_b = A \xrightarrow{m} B; B \xrightarrow{n} C, \\
&\text{G}_c = A \xrightarrow{m} B; C \xrightarrow{n} B, \\
&\text{G}_d = A \xrightarrow{m} B; C \xrightarrow{n} A, \\
&\text{G}_e = A \xrightarrow{m} B; A \xrightarrow{n} B, \\
&\text{G}_f = A \xrightarrow{m} B; C \xrightarrow{n} D.
\end{align*}
\]

**Definition 3.2.5** (Pomsets for parallel compositions [24, 51]). Let \(r = [E, \leq, \lambda]\) and \(r' = [E', \leq', \lambda']\) be two pomsets. The pomset for the parallel composition \(\text{par}(r, r')\) of \(r\) and \(r'\) is defined as

\[
\text{par}(r, r') = [E \sqcup E', (\leq \otimes 1) \cup (\leq' \otimes 2), (\lambda \otimes 1) \cup (\lambda' \otimes 2)].
\]
Roughly speaking, the pomsets for parallel composition \( \text{par}(r, r') \) retains all the causal dependencies from both \( r \) and \( r' \), while does not add any new causal dependencies between \( r \) and \( r' \).

\[
\begin{bmatrix}
AB!m \\
\downarrow \\
AB?m
\end{bmatrix}
\quad
\begin{bmatrix}
AB!n \\
\downarrow \\
AB?n
\end{bmatrix}
\quad
\begin{bmatrix}
AB!m & AB!n \\
\downarrow & \downarrow \\
AB?m & AB?n
\end{bmatrix}
\]

\( G_b = A \xrightarrow{m} B + A \xrightarrow{n} B \)

\( G_p = A \xrightarrow{m} B | A \xrightarrow{n} B \)

Fig. 3.5 Instances of pomsets for branching and parallel compositions

Fig. 3.5 shows two instances of pomsets for a branching composition, a parallel composition and an iteration. The two pomsets at left of the figure describe the pomsets for the branching composition \( G_b = A \xrightarrow{m} B + A \xrightarrow{n} B \), the left and right pomsets correspond to \( A \xrightarrow{m} B \) and \( A \xrightarrow{n} B \) respectively, where the choice is made by participant \( A \). Next, the pomset at right of the figure shows the pomset for the parallel composition \( G_p = A \xrightarrow{m} B | A \xrightarrow{n} B \). The pomset shows the events from the parallel composition components are able to concurrent in each possible order.

Now, we can build the pomsets for the g-choreography describing the on-line shopping application in Fig. 1.4(a) based on the notions reviewed above. The pomsets are shown in Fig. 3.6 where circled numbers are used to refer to the arrows from the following text and two types of arrows distinguish causal dependencies induced by interactions (solid arrows) from those induced by the sequential compositions (dashed arrows). For instance, the solid arrow (1) shows that the send action \( BS!\text{request} \) must happen before the receive action \( BS?\text{request} \) in the interaction \( B \xrightarrow{\text{request}} S \).

Each pomset in Fig. 3.6 corresponds to a branch of the g-choreography in Fig. 1.4(a); the upper pomset corresponds to the choice of the Seller to inform when goods are in the stock while the lower pomset corresponds to the other case.

While, the dashed arrow (2) represents receive action \( BS?\text{request} \) must happen before the send action \( SB!\text{offer} \) since the interaction \( B \xrightarrow{\text{request}} S \) precedes the interaction.
3.3 Well-formedness of g-choreographies

We have reviewed the syntax of g-choreographies and pomsets for global choreographies in last two sections. In this section, we now turn to review the well-formedness of g-choreographies upon pomset semantics.

As mentioned in Fig. 3.3, we have discussed the correctness and in-correctness of branching g-choreographies, we are aware of incorrect g-choreographies may have
3.3 Well-formedness of g-choreographies

potential deadlocks. In order to avoid deadlocks in g-choreographies, it is necessary to
define the correctness of g-choreographies or the well-formedness of g-choreographies.

Then, we give the semantics of g-choreographies based on [24, 51].

**Definition 3.3.1** (Semantics of global choreographies). The semantics of a global
choreography is a family of pomsets defined as

\[
[0] = \{ \epsilon \}
\]

\[
[A \xrightarrow{m} B] = \{\{(e_1, e_2), (e_1, e_1), (e_1, e_2), (e_2, e_2)\}, \lambda\} \text{ where } \lambda = \begin{cases} e_1 \mapsto AB!m \\ e_2 \mapsto AB?m \end{cases}
\]

\[
[G; G'] = \begin{cases} \{\text{seq}(r, r') \mid (r, r') \in [G] \times [G']\} & \text{if } ws(G, G') \\ \bot & \text{otherwise} \end{cases}
\]

\[
[G \mid G'] = \begin{cases} \{\text{par}(r, r') \mid (r, r') \in [G] \times [G']\} & \text{if } wf(G, G') \\ \bot & \text{otherwise} \end{cases}
\]

\[
[G + G'] = \begin{cases} [G] \cup [G'] & \text{if } wb(G, G') \\ \bot & \text{otherwise} \end{cases}
\]

where \( ws(G, G') \), \( wf(G, G') \), \( wb(G, G') \) are called well-sequenced (Definition 3.3.2),
well-forked (Definition 3.3.3) and well-branched (Definition 3.3.4) respectively to check
the meaningfulness of sequential compositions, parallel compositions and branching
compositions of g-choreographies.

We start to elaborate Definition 3.3.1. (1) Obviously, the semantics of 0 is a
singleton set containing \( \epsilon \); (2) the semantics of \( A \xrightarrow{m} B \) shows the send action \( AB!m \)
must happen before the receive action \( AB?m \); (3) the semantics of \( G; G' \) is the set of
all sequential compositions of \( r \) and \( r' \), where \( r \) and \( r' \) are the elements of the pomset
semantics of \( G \) and \( G' \) respectively when the well-sequencedness is satisfied, otherwise,
the semantics of \( G; G' \) is undefined; (4) similarly, the semantics of \( G \mid G' \) is the set of
all parallel compositions of \( r \) and \( r' \), where \( r \) and \( r' \) are the elements of the pomset
semantics of $G$ and $G'$ respectively when the well-forkedness holds, otherwise, the semantics of $G \mid G'$ is undefined; (5) the semantics of $G + G'$ is the union of semantics of $G$ and $G'$ if $G + G'$ is well-branched, otherwise, the semantics of $G + G'$ is undefined. (6) the semantics of $\ast \{G\} \ominus A$ is the set of iteration compositions of $r$, where $r$ is the element of the semantics of $G$, we will elaborate the iteration compositions of g-choreographies in Chapter 5.

From illustrated above, the semantics of empty, interactions, iterations are intuitive from the notion of pomsets. However, sequential compositions, parallel compositions and branching compositions of g-choreographies have three different constraints well-sequencedness $ws(G, G')$, well-forkedness $wf(G, G')$ and well-branchedness $wb(G, G')$ to define their semantics. We are now ready to review these three constraints in the following three subsections.

3.3.1 Well-sequencedness

Now, we review the well-sequencedness $ws(G, G')$ first.

**Definition 3.3.2** (Well-sequenced [24, 51]). Pomsets $r = [\mathcal{E}, \leq, \lambda]$ and $r' = [\mathcal{E}', \leq', \lambda']$ are well sequenced if

$$\leq_{\text{seq}(r, r')} := \{(e \in \mathcal{E} \mid \lambda(e) \in \mathcal{L}' \} \times 1) \times \{(e \in \mathcal{E}' \mid \lambda'(e) \in \mathcal{L}' \} \times 2)\$$

we write $ws(r, r')$ when $r$ and $r'$ are well-sequenced and, for $G, G', ws(G, G')$ when $[G] \neq \bot \land [G'] \neq \bot \land \forall r \in [G], r' \in [G'] : ws(r, r')$.

We consider that sequential compositions of pomsets $r, r'$ are well-sequenced if all receive actions in $r$ happen before any send action in $r'$. In other words, the sequential composition $G; G'$ is well-sequenced if all receive actions in $[G]$ happen before any send action in $[G']$.

Let us review the instances in Fig. 3.7, the sequential composition $G = A \xrightarrow{m} B ; B \xrightarrow{n} C$ is well-sequenced since the receive action $AB?m$ happens before the send action $BC!n$ and the pomset semantics of $A \xrightarrow{m} B$ and $B \xrightarrow{n} C$ are defined. However, the
3.3 Well-formedness of g-choreographies

\[ G = A \xrightarrow{m} B; B \xrightarrow{n} C \]

Fig. 3.7 Instances of well-sequencedness

sequential composition \( G' = A \xrightarrow{m} B; C \xrightarrow{n} D \) is not well-sequenced since there is not such happen-before relationships between the receive action \( AB?m \) and the send action \( CD!n \). Therefore, the semantics of \( G' = A \xrightarrow{m} B; C \xrightarrow{n} D \) has not been defined. From these two instances, we are aware of the well-sequencedness restrict requires sequential relationships between components.

3.3.2 Well-forkedness

Then, we review the well-forkedness \( wf(G, G') \).

**Definition 3.3.3** (Well-forked [24, 51]). Pomsets \( r = [E, \leq, \lambda] \) and \( r' = [E', \leq', \lambda'] \) are well forked if

\[ \lambda(E) \cap \lambda'(E') \cap L^2 = \emptyset \]

we write \( wf(r, r') \) when \( r \) and \( r' \) are well-forked and, for \( G, G', wf(G, G') \) when \( [G] \neq \bot \wedge [G'] \neq \bot \wedge \forall r \in [G], r' \in [G'] : wf(r, r') \).

We consider that parallel compositions of pomsets \( r, r' \) are well-forked if the input events of \( r \) and \( r' \) are disjoint. In other words, the parallel composition \( G; G' \) is well-forked if the input events of \( G \) and \( G' \) are disjoint, as well as \( [G] \) and \( [G'] \) have been both defined.

Let us review the instances in Fig. 3.8, the parallel composition \( G = (A \xrightarrow{m} B; B \xrightarrow{n} C) \mid (A \xrightarrow{m} B; B \xrightarrow{n} C) \) is not well-sequenced since there is same input event (corresponding to \( AB?m \)) in both \( [A \xrightarrow{m} B; B \xrightarrow{n} C] \) and \( [A \xrightarrow{m} B; B \xrightarrow{n} C] \). The circumstance is that if the left thread of \( A \) executes \( AB!m \), then the right thread of \( B \) can execute
3.3 Well-formedness of g-choreographies

3.3.3 Well-branchedness

Next, we turn to review the well-branchedness $\text{wb}(G, G')$.

**Definition 3.3.4** (Well-branched [24, 51]). Two global choreographies $G$ and $G'$ are well-branched when

1. there is one active participant (Definition 3.3.7) in $G + G'$,

2. all the other participants of $G + G'$ are passive participants (Definition 3.3.8).
3.3 Well-formedness of g-choreographies

We write \( wb(G, G') \) are well-branched.

From Definition 3.3.4 above, we are aware of the well-branchedness depends on the definitions of active participants and passive participants.

**Definition 3.3.5.** Let \( r = [\mathcal{E}, \leq, \lambda] \) be a pomset. The pomset

\[
r_{\downarrow A} = [\mathcal{E}_{r,A}, \leq \cap (\mathcal{E}_{r,A}, \mathcal{E}_{r,A}), \lambda \mid_{\mathcal{E}_{r,A}}]
\]

is the pomset of \( A \) in \( r \), where \( \mathcal{E}_{r,A} = \{ e \in \mathcal{E} \mid sbj(e) = A \} \), \( \lambda \mid_{\mathcal{E}_{r,A}} \) denotes the restriction of the function \( \lambda \) to the subset \( \mathcal{E}_{r,A} \) of its domain.

The pomset \( r_{\downarrow A} \) is the projection of participant \( A \) from pomset \( r \). Definition 3.3.5 helps us to analyse the pomset semantics of g-choreographies from the participant perspectives.

As a matter of fact, the behaviours of two branches are able to be same up to a divergence point, where participants start to behave different. The branching start point of global choreography \( G + G' \) has been regarded as the divergence point in well-branchedness. Therefore, according to the divergence point, we define active participants and passive participants in well-branchedness.

**Definition 3.3.6.** Given a branching global graph \( G + G' \), let \( \tilde{l}_1 \) and \( \tilde{l}_2 \) be two subsets of \( \mathcal{L} \), we define \( \text{div}_A(G, G') = (\tilde{l}_1, \tilde{l}_2) \), such that

\[
\tilde{l}_1 = \bigcup_{r \in [G]} \lambda \mid_{\mathcal{E}_{r,A}} (\min r_{\downarrow A}) \quad \text{and} \quad \tilde{l}_2 = \bigcup_{r' \in [G']} \lambda \mid_{\mathcal{E}_{r',A}} (\min r'_{\downarrow A})
\]

where \( \min r = \{ e \in \mathcal{E} \mid \exists e' \in \mathcal{E} \colon e' \neq e \land e' \leq e \} \).

**Definition 3.3.7 (Active participants).** A participant \( A \in \mathcal{P} \) is active in \( G + G' \) if, letting \( \text{div}_A(G, G') = (\tilde{l}_1, \tilde{l}_2) \) we have

\[
\tilde{l}_1 \cup \tilde{l}_2 \subseteq \mathcal{L}^I \quad \tilde{l}_1 \cap \tilde{l}_2 = \emptyset \quad \tilde{l}_1 \neq \emptyset \quad \tilde{l}_2 \neq \emptyset.
\]
An active participant $A$ in $G + G'$ must send different messages to inform other participants from the branching starting point of branching composition of global choreographies.

**Definition 3.3.8** (Passive participants). A participant $A \in \mathcal{P}$ is passive in $G + G'$ if letting $\text{div}_{A}(G, G') = (\tilde{l}_1, \tilde{l}_2)$ we have

$$\tilde{l}_1 \cup \tilde{l}_2 \subseteq \mathcal{C}^2 \quad \tilde{l}_1 \cap \tilde{l}_2 = \emptyset \quad \tilde{l}_1 \neq \emptyset \quad \tilde{l}_2 \neq \emptyset$$

A passive participant $A$ in $G + G'$ must receive messages from the branching starting point of branching composition of global choreographies.

Fig. 3.9 shows seven instances of branching composition of global choreographies below; we analyse if they are well-branched.

![Fig. 3.9 Instances of well-branchedness](image-url)
3.3 Well-formedness of g-choreographies

Instance $G_1$. For participant $A$, we have $\mathcal{R}' = [G']_{A} = \{[AB!m]\}$ and $\mathcal{R}'' = [G'']_{A} = \{[AB!m]\}$, then $\text{div}_A(G', G'') = (\{AB!m\}, \{AB!m\})$. After that, participant $A$ is neither active nor passive since $\{AB!m\} \cup \{AB!m\} \subseteq \mathcal{L}$, $\{AB!m\} \cap \{AB!m\} \neq \emptyset$. For participant $B$, we have $\mathcal{R}' = [G']_{B} = \{[AB?m]\}$ and $\mathcal{R}'' = [G'']_{B} = \{[AB?m]\}$, then $\text{div}_B(G', G'') = (\{AB?m\}, \{AB?m\})$. Then, participant $B$ is neither active nor passive since $\{AB?m\} \cup \{AB?m\} \subseteq \mathcal{L}$, $\{AB?m\} \cap \{AB?m\} \neq \emptyset$. Therefore, $G_1$ has two undefined participants, according to Definition 3.3.4, $G_1$ is not well-branched.

Instance $G_2$. For participant $A$, we have $\mathcal{R}' = [G']_{A} = \{[AB!m]\}$ and $\mathcal{R}'' = [G'']_{A} = \{[AB!n]\}$, then $\text{div}_A(G', G'') = (\{AB!m\}, \{AB!n\})$. After that, participant $A$ is an active participant since $\{AB!m\} \cup \{AB!n\} \subseteq \mathcal{L}$, $\{AB!m\} \cap \{AB!n\} = \emptyset$. For participant $B$, we have $\mathcal{R}' = [G']_{B} = \{[AB?m]\}$ and $\mathcal{R}'' = [G'']_{B} = \{[AB?n]\}$, then $\text{div}_B(G', G'') = (\{AB?m\}, \{AB?n\})$. Then, participant $B$ is a passive participant since $\{AB?m\} \cup \{AB?n\} \subseteq \mathcal{L}$. Therefore, $G_2$ has an active participant and a passive participant, according to Definition 3.3.4, $G_2$ is well-branched.

Instance $G_3$. For participant $A$, we have $\mathcal{R}' = [G']_{A} = \{[AB!m]\}$ and $\mathcal{R}'' = [G'']_{A} = \{[AC!n]\}$, then $\text{div}_A(G', G'') = (\{AB!m\}, \{AC!n\})$. After that, participant $A$ is an active participant since $\{AB!m\} \cup \{AC!n\} \subseteq \mathcal{L}$, $\{AB!m\} \cap \{AB!m\} = \emptyset$. For participant $B$, we have $\mathcal{R}' = [G']_{B} = \{[AB?m]\}$ and $\mathcal{R}'' = [G'']_{B} = \emptyset$, then $\text{div}_B(G', G'') = (\{AB?m\}, \emptyset)$. Then, participant $B$ is neither active nor passive since $\{G'\}_{B} = \emptyset$. For participant $C$, we have $\mathcal{R}' = [G']_{C} = \emptyset$ and $\mathcal{R}'' = [G'']_{C} = \{[AC?n]\}$, then $\text{div}_C(G', G'') = (\emptyset, \{AC?n\})$. Then, participant $C$ is neither active nor passive since $\{G'\}_{C} = \emptyset$. Therefore, $G_3$ has an active participant and two undefined participants, according to Definition 3.3.4, $G_3$ is not well-branched.

Instance $G_4$. For participant $A$, we have $\mathcal{R}' = [G']_{A} = \{[AB!m]\}$ and $\mathcal{R}'' = [G'']_{A} = \emptyset$, then $\text{div}_A(G', G'') = (\{AB!m\}, \emptyset)$. After that, participant $A$ is neither active nor passive since $\{G'\}_{A} = \emptyset$. For participant $B$, we have $\mathcal{R}' = [G']_{B} = \{[AB?m]\}$ and $\mathcal{R}'' = [G'']_{B} = \emptyset$, then $\text{div}_B(G', G'') = (\{AB?m\}, \emptyset)$. Then, participant $B$ is neither active nor passive since $\{G''\}_{B} = \emptyset$. For participant $C$, we have $\mathcal{R}' = [G']_{C} = \emptyset$ and $\mathcal{R}'' = [G'']_{C} = \{[CD!n]\}$, then $\text{div}_C(G', G'') = (\emptyset, \{CD!n\})$. Then, participant $C$ is
neither active nor passive since $[G']_{L}\subseteq\emptyset$. For participant $D$, we have $R' = [G']_{L} = \emptyset$ and $R'' = [G''_{L}] = \{[CD?n]\}$, then $div_D(G',G'') = (\emptyset, \{CD?n\})$. Then, participant $D$ is neither active nor passive since $[G']_{L} = \emptyset$. Therefore, $G_4$ has four undefined participants, according to Definition 3.3.4, $G_4$ is not well-branched.

**Instance** $G_5$. For participant $A$, we have $R' = [G']_{L} = \{[AB!m]\}$ and $R'' = [G''_{L}] = \{[AB!n]\}$, then $div_A(G',G'') = ([AB!m], [AB!n])$. After that, participant $A$ is an active participant since $\{AB!m\} \subseteq L^1$, $\{AB!m\} \cap \{AB!n\} = \emptyset$. For participant $B$, we have $R' = [G']_{L} = \{[AB?m \leftarrow CB?x]\}$ and $R'' = [G''_{L}] = \{[AB?n \rightarrow CB?x]\}$, then $div_B(G',G'') = ([AB?m], [AB?n])$. Then, participant $B$ is a passive participant since $\{AB?m\} \cup \{AB?n\} \subseteq L^2$. For participant $C$, we have $R' = [G']_{L} = \emptyset$ and $R'' = [G''_{L}] = \emptyset$, then $div_C(G',G'') = \emptyset$. Therefore, $G_5$ has an undefined participant, an active participant a passive participant, according to Definition 3.3.4, $G_5$ is not well-branched.

**Instance** $G_6$. For participant $A$, we have $R' = [G']_{L} = \{[AB!m]\}$ and $R'' = [G''_{L}] = \{[AC!n]\}$, then $div_A(G',G'') = ([AB!m], [AC!n])$. After that, participant $A$ is an active participant since $\{AB!m\} \subseteq L^1$, $\{AB!m\} \cap \{AC!n\} = \emptyset$. For participant $B$, we have $R' = [G']_{L} = \{[AB?m \rightarrow BC!x]\}$ and $R'' = [G''_{L}] = \{[AB?n]\}$, then $div_B(G',G'') = ([AB?m], [CB?y])$. Then, participant $B$ is a passive participant since $\{AB?m\} \cup \{CB?y\} \subseteq L^2$. For participant $C$, we have $R' = [G']_{L} = \emptyset$ and $R'' = [G''_{L}] = \emptyset$, then $div_C(G',G'') = \emptyset$. Therefore, $G_6$ has an active participant and two passive participants, according to Definition 3.3.4, $G_6$ is well-branched.

**Instance** $G_7$. For participant $A$, we have $R' = [G']_{L} = \{[AB!m]\}$ and $R'' = [G''_{L}] = \{[AB!n]\}$, then $div_A(G',G'') = ([AB!m], [AB!n])$. After that, participant $A$ is an active participant since $\{AB!m\} \cup \{AB!n\} \subseteq L^1$, $\{AB!m\} \cap \{AB!n\} = \emptyset$. For participant $B$, we have $R' = [G']_{L} = \{[AB?m \rightarrow CB?x]\}$ and $R'' = [G''_{L}] = \{[CB?y \rightarrow AB?n]\}$, then $div_B(G',G'') = ([AB?m], [CB?y])$. Then, participant $B$ is a passive
participant since \( \{AB?m\} \cup \{CB?y\} \subseteq \mathcal{L} \). For participant C, we have \( \mathcal{R}' = \llbracket \mathcal{G}' \rrbracket \mathcal{L}_{C} = \{[CB!x]\} \) and \( \mathcal{R}'' = \llbracket \mathcal{G}'' \rrbracket \mathcal{L}_{C} = \{[CB!y]\} \), then \( \text{div}_C(\mathcal{G}', \mathcal{G}'') = (\{CB!x\}, \{CB!y\}) \). Then, participant C is an active participant since \( \{CB!x\} \cup \{CB!y\} \subseteq \mathcal{L} \), \( \{CB!x\} \cap \{CB!y\} = \emptyset \). Therefore, \( \mathcal{G}_7 \) has two active participants and a passive participant, according to Definition 3.3.4, \( \mathcal{G}_7 \) is not well-branched.

From these instances above, they illustrate defining participants by the divergence point at branching compositions of global choreographies.

### 3.3.4 Another well-branchedness

As we see the well-branchedness in the previous subsection, active participants (Definition 3.3.7) and passive participants (Definition 3.3.8) are defined by the branching start point of global choreography \( \mathcal{G} + \mathcal{G}' \). However, sometimes, the branching start point is not the real divergence point of branching composition of global choreographies. We consider that a real divergence point must explicitly identify where participants start to behave different.

![Diagram](image)

**Fig. 3.10** A general well-branched branching composition

For instance, Fig. 3.10 shows a branching composition \( \mathcal{G} = A \xrightarrow{m} B; C \xrightarrow{n} D; A \xrightarrow{m} B \). According to Definition 3.3.4, \( \mathcal{G} \) is not well-branched since participants \( A, B, C \) and \( D \) are all undefined participants. Participant \( A \) behaves \( AB!m \) in both branches at the branching start point, but we can treat \( AB!m \) is a common part of the
branches for which participants behave uniformly in both branches of the choice. After this common part, we consider it is the real divergence point of participant $A$. And participant $A$ behaves nothing after the real divergence point. Similarly for participants $B$, $C$ and $D$. Then, we start to review the well-branchedness of branching compositions based on the real divergence point. In order to distinguish the well-branchedness (Definition 3.3.4), we call it general well-branchedness.

**Definition 3.3.9** (General well-branched [24, 51]). *Two global choreographies $G$ and $G'$ are general well-branched when*

1. there is at most one general active participant (Definition 3.3.14) in $G + G'$,

2. all the other participants of $G + G'$ are general passive participants (Definition 3.3.15).

We write $gwb(G, G')$ are general well-branched.

In order to identify the common part and the real divergence point between branches of a participant, we define the prefix and suffix of pomsets according to [24, 51]. We let $\_ | X$ denote the restriction of a function to a subset $X$ of its domain.

**Definition 3.3.10.** A pomset $r' = [E', \leq', \lambda']$ is a sub-pomset of pomset $r = [E, \leq, \lambda]$ if

\[
E' \subseteq E \quad \text{and} \quad \leq' = \leq \cap (E' \times E') \quad \text{and} \quad \lambda' = \lambda |_{E'}
\]

A sub-pomset $r'$ of $r$ is a prefix of $r$ if $\leq \cap ((E' \setminus E) \times E') = \emptyset$.

**Definition 3.3.11.** The suffix of a pomset $r$ with respect to one of its prefixes $r'$, denoted as $r - r'$, is the pomset $[E', \leq_r \cap (E' \times E'), \lambda_r |_{E'}]$, where $E' = E_r \setminus E_{r'}$.

Based on the above definitions of prefix and suffix of pomsets, we define the common part (prefix) and the real divergence point [24, 51].

**Definition 3.3.12.** The pair of functions $(\phi, \psi)$ is an A-prefix map of $G_1$ and $G_2$ if
• dom \( \phi \) and cod \( \psi \) are partitions of \([G_1]_{\downarrow A}\) and \([G_2]_{\downarrow A}\) respectively, \( \phi \) is bijective and

• dom \( \psi = \) dom \( \phi \) and cod \( \psi \) is a set of pomsets

such that, for all \( R \in \) dom \( \phi \) and \( (r, r') \in R \times \phi(R) \), the pomset \( \psi(R) \) is a prefix pomset of both \( r \) and \( r' \).

**Definition 3.3.13.** Let \( \tilde{l}_1 \) and \( \tilde{l}_2 \) be two subsets of \( \mathcal{L} \), we say that \((\tilde{l}_1, \tilde{l}_2)\) is the divergence point of \( G_1 \) and \( G_2 \) (or that \( G_1 \) and \( G_2 \) diverge at \((\tilde{l}_1, \tilde{l}_2)\)) with respect to the A-prefix map \((\phi_A, \psi_A)\), denoted \( \text{div}^{\phi_A, \psi_A} = (\tilde{l}_1, \tilde{l}_2) \) if

\[
\tilde{l}_1 = \bigcup_{R \in \text{dom} \phi_A, r \in R} \lambda_r(\min(r - \psi_A(R))) \setminus \tilde{l}_p
\]

\[
\tilde{l}_2 = \bigcup_{R \in \text{cod} \psi_A, r \in R} \lambda_r(\min(r - \psi_A(\phi_A^{-1}(R)))) \setminus \tilde{l}_p
\]

where

\[
\tilde{l}_p = \bigcup_{B \neq A, R' \in \text{dom} \phi_B, r' \in R'} \text{dual}(\psi_B(R')).
\]

For a pomset \( r \), \( \min r = \{ e \in E_r \mid \exists e' \in E_r : e' \neq e \land e' \leq r, e \} \). The operation \( \text{dual} \) returns the set of the dual events’ labels.

The bijective function \( \phi \) reflects the pomsets both in \([G']_{\downarrow A}\) and \([G'']_{\downarrow A}\) which have an common behaviour determined by the prefix map \( \psi \). Therefore, \( \tilde{l}_1 \) and \( \tilde{l}_2 \) are the union of the minimal events of the pomsets in \([G']_{\downarrow A}\) and \([G'']_{\downarrow A}\) after the real divergence points.

**Definition 3.3.14** (General active participants). A participant \( A \in \mathcal{P} \) is general active in \( G + G' \) if there exists an A-prefix map \((\phi, \psi)\) of \( G_1 \) and \( G_2 \) such that, if \( \text{div}^{\phi, \psi}(G, G') = (\tilde{l}_1, \tilde{l}_2) \) holds, where

\[
\tilde{l}_1 \cup \tilde{l}_2 \subseteq \mathcal{L}' \quad \tilde{l}_1 \cap \tilde{l}_2 = \emptyset \quad \tilde{l}_1 \neq \emptyset \quad \tilde{l}_2 \neq \emptyset.
\]
Definition 3.3.15 (General passive participants). A participant \( A \in \mathcal{P} \) is general passive in \( G + G' \) if there exists an \( A \)-prefix map \( (\phi, \psi) \) of \( G_1 \) and \( G_2 \) such that, if \( \text{div}_A^{\phi,\psi}(G, G') = (\hat{I}_1, \hat{I}_2) \) holds, where

- \( \hat{I}_1 \cup \hat{I}_2 \subseteq \mathcal{L}^2 \) and \( \hat{I}_1 = \emptyset \iff \hat{I}_2 = \emptyset \)
- \( \forall R \in \text{dom } \phi, r \in R : \hat{I}_2 \cap \lambda_r(\varepsilon_{r-\psi(R)}) = \emptyset \)
- \( \forall R \in \text{cod } \phi, r \in R : \hat{I}_1 \cap \lambda_r(\varepsilon_{r-\psi(\phi^{-1}(R))}) = \emptyset \)

Then, let us review the branching composition \( G = A \xrightarrow{m} B; C \xrightarrow{n} D + C \xrightarrow{n} D; A \xrightarrow{m} B \) in Fig. 3.10. As we say participants \( A, B, C, D \) do not behave anything from real divergence points, so all of them are defined as general passive participants. Therefore, the branching composition \( G \) is general well-branched. In other words, due to participants \( A, C \) sending the same messages to participants \( B, D \) respectively in both branches, therefore, participants \( A, B, C \) and \( D \) behave nothing after prefix maps removing all the common parts.

In addition, let us recall the instances shown in Fig. 3.9 on page 30 to illustrate the real divergence points as well as the general well-branchedness.

**Instance \( G_1 \).** For participant \( A \), we have \( R' = \llbracket G' \rrbracket_{\downarrow A} = \{ [AB!m] \} \) and \( R'' = \llbracket G'' \rrbracket_{\downarrow A} = \{ [AB!m] \} \). Let \( \phi = R' \mapsto R'' \) and \( \psi = R' \mapsto \epsilon \), that means the empty prefix, then the divergence point is \( \text{div}_A^{\phi,\psi}(G', G'') = (\{ AB!m \}, \{ AB!m \}) \). However, a different divergence point can be computed by using prefix \( \{ AB!m \} \), let \( \psi = R' \mapsto [AB!m] \), then \( \text{div}_A^{\phi,\psi}(G', G'') = (\emptyset, \emptyset) \). After that, participant \( A \) is general passive participant. Similarly, for participant \( B \), we have \( R' = \llbracket G' \rrbracket_{\downarrow B} = \{ [AB?m] \} \) and \( R'' = \llbracket G'' \rrbracket_{\downarrow B} = \{ [AB?m] \} \), for the non-empty prefix, let \( \phi = R' \mapsto R'' \) and \( \psi = R' \mapsto [AB?m] \), then \( \text{div}_B^{\phi,\psi}(G', G'') = (\emptyset, \emptyset) \). Then, participant \( B \) is a general passive participant. Therefore, \( G_1 \) has two general passive participants, according to Definition 3.3.9, \( G_1 \) is general well-branched.

**Instance \( G_2 \).** For participant \( A \), we have \( R' = \llbracket G' \rrbracket_{\downarrow A} = \{ [AB!m] \} \) and \( R'' = \llbracket G'' \rrbracket_{\downarrow A} = \{ [AB!n] \} \). Let \( \phi = R' \mapsto R'' \) and \( \psi = R' \mapsto \epsilon \), then the divergence point \( \text{div}_A^{\phi,\psi}(G', G'') = (\{ AB!m \}, \{ AB!n \}) \). After that, participant \( A \) is a general active
3.3 Well-formedness of g-choreographies

participant. Similarly, for participant $B$, we have $\mathcal{R}' = [G']_{\downarrow B} = \{ [AB?m] \}$ and $\mathcal{R}'' = [G'']_{\downarrow B} = \{ [AB?n] \}$. Let $\phi = \mathcal{R}' \mapsto \mathcal{R}''$ and $\psi = \mathcal{R}' \mapsto \emptyset$, then $\text{div}_B^{\phi,\psi}(G',G'') = (\{ AB?m \}, \{ AB?n \})$. Then, participant $B$ is a general passive participant. Therefore, $G_2$ has a general active participant and a general passive participant, according to Definition 3.3.9, $G_2$ is general well-branched.

**Instance $G_3$.** For participant $A$, we have $\mathcal{R}' = [G']_{\downarrow A} = \{ [AB!m] \}$ and $\mathcal{R}'' = [G'']_{\downarrow A} = \{ [AC!n] \}$. Let $\phi = \mathcal{R}' \mapsto \mathcal{R}''$ and $\psi = \mathcal{R}' \mapsto \emptyset$, then the divergence point $\text{div}_A^{\phi,\psi}(G',G'') = (\{ AB!m \}, \{ AC!n \})$. After that, participant $A$ is a general active participant. For participant $B$, we have $\mathcal{R}' = [G']_{\downarrow B} = \{ [AB?m] \}$ and $\mathcal{R}'' = [G'']_{\downarrow B} = \emptyset$. Let $\phi = \mathcal{R}' \mapsto \mathcal{R}''$ and $\psi = \mathcal{R}' \mapsto \epsilon$, then $\text{div}_B^{\phi,\psi}(G',G'') = (\{ AB?m \}, \emptyset)$. Therefore, participant $B$ is neither general active nor general passive. For participant $C$, we have $\mathcal{R}' = [G']_{\downarrow C} = \emptyset$ and $\mathcal{R}'' = [G'']_{\downarrow C} = \{ [AC?n] \}$. Let $\phi = \mathcal{R}' \mapsto \mathcal{R}''$ and $\psi = \mathcal{R}' \mapsto \epsilon$, then $\text{div}_C^{\phi,\psi}(G',G'') = (\emptyset, \{ AC?n \})$. So that, participant $C$ is neither general active nor general passive. Therefore, $G_3$ has a general active participant and two undefined participants, according to Definition 3.3.9, $G_3$ is not general well-branched.

**Instance $G_4$.** For participant $A$, we have $\mathcal{R}' = [G']_{\downarrow A} = \{ [AB!m] \}$ and $\mathcal{R}'' = [G'']_{\downarrow A} = \emptyset$. Let $\phi = \mathcal{R}' \mapsto \mathcal{R}''$ and $\psi = \mathcal{R}' \mapsto \epsilon$, then $\text{div}_A^{\phi,\psi}(G',G'') = (\{ AB!m \}, \emptyset)$. After that, participant $A$ is neither general active nor general passive. For participant $B$, we have $\mathcal{R}' = [G']_{\downarrow B} = \{ [AB?m] \}$ and $\mathcal{R}'' = [G'']_{\downarrow B} = \emptyset$. Let $\phi = \mathcal{R}' \mapsto \mathcal{R}''$ and $\psi = \mathcal{R}' \mapsto \epsilon$, then $\text{div}_B^{\phi,\psi}(G',G'') = (\{ AB?m \}, \emptyset)$. Then, participant $B$ is neither general active nor general passive. For participant $C$, we have $\mathcal{R}' = [G']_{\downarrow C} = \emptyset$ and $\mathcal{R}'' = [G'']_{\downarrow C} = \{ [CD!n] \}$. Let $\phi = \mathcal{R}' \mapsto \mathcal{R}''$ and $\psi = \mathcal{R}' \mapsto \epsilon$, then $\text{div}_C^{\phi,\psi}(G',G'') = (\emptyset, \{ CD!n \})$. Then, participant $C$ is neither general active nor general passive. For participant $D$, we have $\mathcal{R}' = [G']_{\downarrow D} = \emptyset$ and $\mathcal{R}'' = [G'']_{\downarrow D} = \{ [CD?n] \}$. Let $\phi = \mathcal{R}' \mapsto \mathcal{R}''$ and $\psi = \mathcal{R}' \mapsto \epsilon$, then $\text{div}_D(G',G'') = (\emptyset, \{ CD?n \})$. Then, participant $D$ is neither general active nor general passive. Therefore, $G_4$ has four undefined participants, according to Definition 3.3.9, $G_4$ is not general well-branched.

**Instance $G_5$.** For participant $A$, we have $\mathcal{R}' = [G']_{\downarrow A} = \{ [AB!m] \}$ and $\mathcal{R}'' = [G'']_{\downarrow A} = \{ [AB!n] \}$. Let $\phi = \mathcal{R}' \mapsto \mathcal{R}''$ and $\psi = \mathcal{R}' \mapsto \epsilon$, then the divergence point
\[ \text{div}^\phi_\psi(G', G^\prime) = ([AB!m], \{AB!n\}). \] After that, participant A is a general active participant. For participant B, we have \( R' = [G]_{\downarrow B} = \{[AB?m \mapsto CB?x]\} \) and \( R'' = [G^\prime]_{\downarrow B} = \{[AB?n \mapsto CB?x]\} \). Let \( \phi = R' \rightarrow R'' \) and \( \psi = R' \rightarrow \epsilon \), then \( \text{div}^\phi_\psi(G', G^\prime) = ([AB?m], \{AB!n\}) \). So participant B is a general passive participant. For participant C, we have \( R' = [G]_{\downarrow C} = \{[CB!x]\} \) and \( R'' = [G^\prime]_{\downarrow C} = \{[CB!x]\} \). Let \( \phi = R' \rightarrow R'' \) and \( \psi = R' \rightarrow \epsilon \), then \( \text{div}^\phi_\psi(G', G^\prime) = (\emptyset, \emptyset) \). Then, participant C is a general passive participant. Therefore, \( G_5 \) has a general active participant and two general passive participants, according to Definition 3.3.9, \( G_5 \) is general well-branched.

**Instance \( G_6 \).** For participant A, we have \( R' = [G]_{\downarrow A} = \{[AB!m]\} \) and \( R'' = [G^\prime]_{\downarrow A} = \{[AC!n]\} \). Let \( \phi = R' \rightarrow R'' \) and \( \psi = R' \rightarrow \epsilon \), then \( \text{div}^\phi_\psi(G', G^\prime) = ([AB!m], \{AC!n\}) \). After that, participant A is a general active participant. For participant B, we have \( R' = [G]_{\downarrow B} = \{[AB?m \mapsto BC!x]\} \) and \( R'' = [G^\prime]_{\downarrow B} = \{[CB?y]\} \). Let \( \phi = R' \rightarrow R'' \) and \( \psi = R' \rightarrow \epsilon \), then \( \text{div}^\phi_\psi(G', G^\prime) = (\emptyset, \emptyset) \). So participant C is a general passive participant. Therefore, \( G_6 \) has a general active participant and two general passive participants, according to Definition 3.3.9, \( G_6 \) is general well-branched.

**Instance \( G_7 \).** For participant A, we have \( R' = [G]_{\downarrow A} = \{[AB!m]\} \) and \( R'' = [G^\prime]_{\downarrow A} = \{[AB!n]\} \). Let \( \phi = R' \rightarrow R'' \) and \( \psi = R' \rightarrow \epsilon \), then \( \text{div}^\phi_\psi(G', G^\prime) = ([AB!m], \{AB!n\}) \). After that, participant A is a general active participant. For participant B, we have \( R' = [G]_{\downarrow B} = \{[AB?m \mapsto CB?x]\} \) and \( R'' = [G^\prime]_{\downarrow B} = \{[CB?y \mapsto AB?n]\} \). Let \( \phi = R' \rightarrow R'' \) and \( \psi = R' \rightarrow \epsilon \), then \( \text{div}^\phi_\psi(G', G^\prime) = (\emptyset, \emptyset) \). Then, participant B is a general passive participant. For participant C, we have \( R' = [G]_{\downarrow C} = \{[CB!x]\} \) and \( R'' = [G^\prime]_{\downarrow C} = \{[CB!y]\} \). Let \( \phi = R' \rightarrow R'' \) and \( \psi = R' \rightarrow \epsilon \), then \( \text{div}^\phi_\psi(G', G^\prime) = ([CB!x], \{CB!y\}) \). So participant C is a general active participant. Therefore, \( G_7 \) has two general active participants and a general passive participant, according to Definition 3.3.9, \( G_7 \) is not general well-branched.
The instances above show the general well-branchedness upon the real divergence points of branching composition of global choreographies. And we are aware of the importance of prefix map in finding real divergence points since prefix maps detect the states where the participants start to behave different.

### 3.4 Interface automata

Interface automata [13] play an important role in component-based design and component-based verification. It is a class of automata which could capture input/output behaviours of components in system design and verification. Interfaces are the essential characteristics of this class of automata since they not only specify the interactions with the environments, but also constraint the environments [14].

**Definition 3.4.1** (Interface automata [13]). An interface automaton $M = (V, v^{init}, A, T)$ is a 4-tuple defined as follows,

1. $V$ is a finite set of states,
2. $v^{init} \in V$ is the initial state,
3. $A = A_I \cup A_O \cup A_H$ is the set of all actions, where $A_I$, $A_O$ and $A_H$ are mutually disjoint sets which represent the input, output and internal (or hidden) actions, respectively,
4. $T \subseteq V \times A \times V$ is a set of transitions.

We write $v \xrightarrow{a} v'$ instead of $(v, a, v') \in T$ when the set of transitions $T$ is clear from the context. In addition, interface automata are similar to input/output automata [38, 32], the only difference is that interface automata do not need to be input-enabled, whereas input/output automata are input-enabled in each state.

An interface automaton consists of a set of finite states; an initial state; a set of actions including input actions, output actions and internal actions; a set of transitions.
For instance, Fig. 3.11 shows an interface automaton

\[ B = (V, v^{\text{init}}, A, \mathcal{T}) \]

where

1. \( V = \{ v_0, v_1, v_2, v_3, v_4, v_5, v_6 \} \)
2. \( v^{\text{init}} = v_0 \)
3. \( A = \{ \text{request!}, \text{offer?}, \text{pay!}, \text{notAvailable?}, \epsilon, \text{delivery?} \} \)

4. \( \mathcal{T} = \{ v_0 \overset{\text{request!}}{\rightarrow} v_1, v_1 \overset{\text{offer?}}{\rightarrow} v_2, v_2 \overset{\text{pay!}}{\rightarrow} v_3, v_3 \overset{\epsilon}{\rightarrow} v_4, v_4 \overset{\text{delivery?}}{\rightarrow} v_5, v_5 \overset{\text{notAvailable?}}{\rightarrow} v_6, v_6 \overset{\epsilon}{\rightarrow} v_5 \} \)

The outside rectangle specifies the scope of the interface automaton \( B \), the arrows on the border of rectangle are the interfaces, the leading (resp. leaving) arrow indicates input (resp. output). As well as the exclamation mark ! is used to describe the output/send actions, the question mark ? is used to describe the input/receive actions, and the semicolon ; is used to describe the internal actions which is independent of the effect of the environment. Therefore, the action \( \text{request!} \) of the transition \( v_0 \overset{\text{request!}}{\rightarrow} v_1 \) means that the interface automaton sends \( \text{request} \) to the environment via the output interface \( \text{request} \). Similarly, the action \( \text{offer?} \) of the transition \( v_1 \overset{\text{offer?}}{\rightarrow} v_2 \) means that the interface automaton receives \( \text{offer} \) from the environment via the input interface \( \text{offer} \). Particularly, the action \( \epsilon \) of the transition \( v_3 \overset{\epsilon}{\rightarrow} v_4 \) means that this interface automaton executes an internal action \( \epsilon \) which is not observed by environments and completely irrelevant to the environments.
As mentioned above, interface automata are able to interact with their environments. In fact, environments are interface automata as well, they are able to interact with each other by synchronous communications. An output from an interface automaton must be consumed immediately by the corresponding input from environments during the communication.

**Definition 3.4.2** (Composability [13]). Two interface automata $M$ and $N$ are composable if

\[ A^H_M \cap A_N = \emptyset, \quad A^I_M \cap A^I_N = \emptyset, \]
\[ A^O_M \cap A^O_N = \emptyset, \quad A_M \cap A^H_N = \emptyset. \]

We let $\text{shared}(M,N) = A_M \cap A_N$. If two interface automata $M$ and $N$ are composable, then $\text{shared}(M,N) = (A^I_M \cap A^O_N) \cup (A^O_M \cap A^I_N)$. Intuitively, if two interface automata have some shared or dual interfaces (for instance, $a?$, $a!$), then, they are able to interact with each other via these shared interfaces.

**Definition 3.4.3** (Product [13]). If two interface automata $M$ and $N$ are composable, their product $M \otimes N$ is the interface automaton defined as follows,

\[
V_{M\otimes N} = V_M \times V_N, \\
v^{\text{init}}_{M\otimes N} = (v^{\text{init}}_M, v^{\text{init}}_N), \\
A^I_{M\otimes N} = (A^I_M \cup A^I_N) \setminus \text{shared}(M,N), \\
A^O_{M\otimes N} = (A^O_M \cup A^O_N) \setminus \text{shared}(M,N), \\
A^H_{M\otimes N} = A^H_M \cup A^H_N \cup \text{shared}(M,N).
\]
The set \( T_{M \otimes N} \) is defined by

\[
T_{M \otimes N} = \left\{ (v,u) \stackrel{\alpha}{\rightarrow} (v',u) \mid (v \stackrel{\alpha}{\rightarrow} v') \in T_M \land \alpha \notin \text{shared}(M,N) \land u \in V_N \right\} \\
\cup \left\{ (v,u) \stackrel{\alpha}{\rightarrow} (v,u') \mid (u \stackrel{\alpha}{\rightarrow} u') \in T_N \land \alpha \notin \text{shared}(M,N) \land v \in V_M \right\} \\
\cup \left\{ (v,u) \stackrel{a_1}{\rightarrow} (v',u') \mid (v \stackrel{a_1}{\rightarrow} v') \in T_M \land (u \stackrel{a_2}{\rightarrow} u') \in T_N \land a_1, a_2 \in \text{shared}(M,N) \right\} \\
\cup \left\{ (v,u) \stackrel{a_1}{\rightarrow} (v',u') \mid (v \stackrel{a_2}{\rightarrow} v') \in T_M \land (u \stackrel{a_1}{\rightarrow} u') \in T_N \land a_1, a_2 \in \text{shared}(M,N) \right\}.
\]

As a matter of fact, the binary operation \( \_ \otimes \_ \) are commutative and associative. In order to show these properties, following [43, 46], we define the strong bisimulation between two interface automata.

**Definition 3.4.4** (Strong bisimulation between two interface automata). Given two interface automata \( M \) and \( N \), a binary relation \( S \subseteq V_M \times V_N \) is a bisimulation if \((v_M, v_N) \in S\) implies, for all \( \alpha \in A_M \cup A_N \),

1. if \( v_M \stackrel{\alpha}{\rightarrow} u_M \in T_M \), then, \( \exists u_N : v_N \stackrel{\alpha}{\rightarrow} u_N \in T_N \land (u_M, u_N) \in S \);
2. if \( v_N \stackrel{\alpha}{\rightarrow} u_N \in T_N \), then, \( \exists u_M : v_M \stackrel{\alpha}{\rightarrow} u_M \in T_M \land (u_M, u_N) \in S \).

For convenient, we write \( M = N \) if there is a strong bisimulation between two interface automata \( M \) and \( N \). Then, we show the theorem of the commutativity and associativity of \( \_ \otimes \_ \) operation. We skip the proof here since the proof of Theorem 3.4.5 is similar with the proof of Theorem 3.4.10.

**Theorem 3.4.5** (Commutativity and associativity of product). Let \( M \), \( N \) and \( K \) be pairwise composable interface automata. Then

\[
M \otimes N = N \otimes M, \\
(M \otimes K) \otimes N = M \otimes (N \otimes K).
\]

From Definition 3.4.3, we are aware that the product of two interface automata generates a new interface automaton.
Fig. 3.12 The product of three composable interface automata

Fig. 3.12 demonstrates how interface automata interact with each other. $M$, $N$, $K$ are three mutually composable interface automata. $M$ has an output interface $a!$ where its corresponding input interface $a?$ is in $N$, $N$ has an output interface $b!$ where its corresponding input interface $b?$ is in $K$. Due to the output interface $a!$ from $M$ and the input interface $a?$ from $N$ being shared, in their product $M \otimes N$, an internal action $a;$ is yielded by the synchronization of the output $a!$ and the input $a?$. While the output interface $b!$ in $N$ does not have the corresponding input interface $b?$ in $M$, therefore, the output interface $b!$ will retain without any synchronizations in $M \otimes N$. Similarly for $M \otimes N \otimes K$. From the product $M \otimes N \otimes K$, we are able to see all the shared interfaces among $M$, $N$, $K$ have been synchronized to yield the internal interfaces. Additionally, due to the product operation being commutative and associative, the orders of computing will not effect the product.

**Definition 3.4.6 (Error states [13]).** The set of error states of the product of two composable interface automata $M$ and $N$ is defined by

$$
\text{Error}(M,N) = \{(v,u) \in V_M \times V_N | \begin{align*}
(\exists a \in \text{shared}(M,N) \land a \in A^O_M : (v \xrightarrow{a} v') \in T_M \land a \in A^I_N : (u \xrightarrow{a} u') \notin T_N) \lor \\
(\exists a \in \text{shared}(M,N) \land a \in A^I_N : (u \xrightarrow{a} u') \in T_N \land a \in A^O_M : (v \xrightarrow{a} v') \notin T_M)\}.
\end{align*}
$$
As a matter of fact, the interaction of two interface automata may lead to error states which may introduce deadlocks. Fig. 3.13 shows an example; the red boxed state $v_0u_0$ (simply for $(v_0, u_0)$) in $M \otimes N$ is an error state. Because the state $v_0$ in $M$ has an output $a!$ in the shared interface of $M$ and $N$, but the corresponding state $u_0$ in $N$ does not have a corresponding input $a?$ in the shared interface. As a result, $a$ will never be consumed forever. Therefore, we call the state $v_0u_0$ error states.

![Fig. 3.13 Instance of error states in the product](image)

Then, we review how to deal with error states in interface automata. The idea is to let the execution of interface automata avoid to reach error states as much as possible.

**Definition 3.4.7** (Incompatible states and compatible states [13]). The set of incompatible states of the $\otimes$-product of two interface automata $M$ and $N$ is the smallest set named $\text{Incomp}(M, N)$, such that,

$$\text{Error}(M, N) \subseteq \text{Incomp}(M, N),$$

and $(v', u') \in \text{Incomp}(M, N)$

if $\forall ((v, u) \xrightarrow{a} (v', u')) \in T_{M\otimes N}$ and $(v', u') \in \text{Incomp}(M, N)$, then $(v, u) \in \text{Incomp}(M, N)$.

The set of compatible states of the product of two interface automata $M$ and $N$ is defined as follow,

$$\text{Comp}(M, N) = V_{M\otimes N} \setminus \text{Incomp}(M, N).$$
Definition 3.4.8 (Compatibility [13]). Two interface automata $M$ and $N$ are compatible if $v_{\text{init}M} \otimes v_{\text{init}N} \notin \text{Incomp}(M,N)$, otherwise $M$ and $N$ are incompatible.

Definition 3.4.9 (Composition [13]). If two interface automata $M$ and $N$ are compatible, their composition $M \parallel N$ is the interface automata which is defined as follows,

\begin{align*}
V_{M\parallel N} &= \text{Comp}(M,N), \\
v^{\text{init}}_{M\parallel N} &= (v^{\text{init}}_M, v^{\text{init}}_N), \\
A_{\text{I}}^{\parallel M\parallel N} &= A_{\text{I}}^M \otimes A_{\text{I}}^N, \\
A_{\text{O}}^{\parallel M\parallel N} &= A_{\text{O}}^M \otimes A_{\text{O}}^N, \\
A_{\text{H}}^{\parallel M\parallel N} &= A_{\text{H}}^M \otimes A_{\text{H}}^N, \\
T_{M\parallel N} &= T_M \otimes T_N \cap (\text{Comp}(M,N) \times A_{M\parallel N} \times \text{Comp}(M,N)).
\end{align*}

The new binary operation between two interface automata is called composition ($\_ \parallel \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_}\]
Fig. 3.14 Instances of removing incompatible states and error states

well since $v'_1u'_1$ does not have another successor different from the incompatible state $v'_4u'_4$. Therefore, the composition $M' \parallel N'$ can be constructed by removing $v'_1u'_1$, $v'_2u'_2$ and all their successors.

Similar with the product operations of interface automata, the composition operations of interface automata are also commutative and associative.

**Theorem 3.4.10** (Commutativity and associativity of composition [16]). Let $M$, $N$ and $K$ be pairwise composable interface automata. Then

$$M \parallel N = N \parallel M,$$

$$(M \parallel K) \parallel N = M \parallel (N \parallel K).$$
Proof. In order to prove commutativity of composition, we first show

\[ M \parallel N = N \parallel M. \]

Let \( M = (V_M, v_M^{\text{init}}, A_M, T_M) \) and \( N = (V_N, v_N^{\text{init}}, A_N, T_N) \) be two pairwise composable interface automata. We define a binary relation \( S_1 \subseteq V_M \parallel N \times V_N \parallel M \), where \( S_1 = \{(v_M, v_N), (v_N, v_M)\} \mid v_M \in V_M, v_N \in V_N\} \).

If \((v_M, v_N), (v_N, v_M)\) \in \( S_1 \), for all \( \alpha \in A_M \parallel N \), let \( (v_M, v_N) \xrightarrow{\alpha} (u_M, u_N) \) be a transition in \( M \parallel N \). By the definition of \( M \parallel N \), we have 4 cases:

1. \( v_M \xrightarrow{\alpha} u_M \) and \( v_N = u_N \),
2. \( v_N \xrightarrow{\alpha} u_N \) and \( v_M = u_M \),
3. \( v_M \xrightarrow{a!} u_M \) and \( v_N \xrightarrow{a?} u_N \), \( \alpha = a; \),
4. \( v_M \xrightarrow{a?} u_M \) and \( v_N \xrightarrow{a!} u_N \), \( \alpha = a; \).

In case (1), by the definition of \( M \parallel N \), we have \( (v_N, v_M) \xrightarrow{\alpha} (u_N, u_M) \), and similarly in case (2), case (3), case (4).

Vice versa, let \( (v_N, v_M) \xrightarrow{\alpha} (u_N, u_M) \) be a transition in \( N \parallel M \), we have \( (v_M, v_N) \xrightarrow{\alpha} (u_M, u_N) \) in \( M \parallel N \).

Obviously, we are able to see \( (u_M, u_N), (u_N, u_M) \) \in \( S_1 \). Therefore, according to Definition 3.4.4, the binary relation \( S_1 \) is a strong bisimulation. So that, we prove

\[ M \parallel N = N \parallel M. \]
In order to prove associativity of composition

\[(M \parallel K) \parallel N = M \parallel (N \parallel K),\]

we first show

\[(M \parallel N) \parallel K = M \parallel (N \parallel K).\]

Let \( M = (V_M, v^\text{init}_M, A_M, T_M), N = (V_N, v^\text{init}_N, A_N, T_N) \) and \( K = (V_K, v^\text{init}_K, A_K, T_K) \) be pairwise composable interface automata. We define a binary relation \( S_2 \subseteq V_{(M\parallel N)\parallel K} \times V_{M\parallel (N\parallel K)}, \) where \( S_2 = \{((v_M, v_K), (v_M, (v_N, v_K))) \mid v_M \in V_M, v_N \in V_N, v_K \in V_K\} \).

If \( ((v_M, v_N), (v_M, (v_N, v_K))) \in S_2 \), for all \( \alpha \in A_{(M\parallel N)\parallel K} (A_{M\parallel (N\parallel K)} = A_{M\parallel (N\parallel K)}), \) then, we are going to show \( S_2 \) is a strong bisimulation. Let

\[ ((v_M, v_N), v_K) \xrightarrow{\alpha} ((u_M, u_N), u_K) \]

be a transition in \( (M \parallel N) \parallel K \). By the definition of \( (M \parallel N) \parallel K \), we have 9 cases in total:

1. When \( (v_M, v_N) \xrightarrow{\alpha} (u_M, u_N) \) and \( v_K = u_K, \)
   
   (a) if \( v_M \xrightarrow{\alpha} u_M \) and \( v_N = u_N \), in this case, \( \alpha \) is not a shared interface among \( M, N \) and \( K \), by the definition of \( N \parallel K \), we have \( (v_N, v_K) = (u_N, u_K) \). Then, by the definition of \( M \parallel (N \parallel K) \), we have
   
   \[ (v_M, (v_N, v_K)) \xrightarrow{\alpha} (u_M, (u_N, u_K)), \]

   (b) if \( v_M = u_M \) and \( v_N \xrightarrow{\alpha} u_N \), in this case, \( \alpha \) is not a shared interface among \( M, N \) and \( K \), by the definition of \( N \parallel K \), we have \( (v_N, v_K) \xrightarrow{\alpha} (u_N, u_K) \). Then, by the definition of \( M \parallel (N \parallel K) \), we have
   
   \[ (v_M, (v_N, v_K)) \xrightarrow{\alpha} (u_M, (u_N, u_K)), \]
(c) if $v_M \xrightarrow{a!} u_M$, $v_N \xrightarrow{a?} u_N$ and $\alpha = a!$, in this case, $a!$ is not a shared interface between $N$ and $K$, by the definition of $N \parallel K$, we have $(v_N, v_K) \xrightarrow{a?} (u_N, u_K)$; $a!$, $a?$ are shared interfaces between $M$ and $N \parallel K$, by the definition of $M \parallel (N \parallel K)$, we have

$$(v_M, (v_N, v_K)) \xrightarrow{\alpha} (u_M, (u_N, u_K)),$$

(d) if $v_M \xrightarrow{a?} u_M$, $v_N \xrightarrow{a!} u_N$ and $\alpha = a!$, in this case, $a!$ is not a shared interface between $N$ and $K$, by the definition of $N \parallel K$, we have $(v_N, v_K) \xrightarrow{a!} (u_N, u_K)$; $a!$, $a?$ are shared interfaces between $M$ and $N \parallel K$, by the definition of $M \parallel (N \parallel K)$, we have

$$(v_M, (v_N, v_K)) \xrightarrow{\alpha} (u_M, (u_N, u_K)),$$

2. when $(v_M, v_N) = (u_M, u_N)$ and $v_K \xrightarrow{\alpha} u_K$, in this case, $\alpha$ is not a shared interface among $M$, $N$ and $K$, by the definition of $M \parallel N$, we have $v_M = u_M$, $v_N = u_N$. Then, by the definition of $N \parallel K$, we have $(v_N, v_K) \xrightarrow{\alpha} (u_N, u_K)$. After that, by the definition of $M \parallel (N \parallel K)$, we have

$$(v_M, (v_N, v_K)) \xrightarrow{\alpha} (u_M, (u_N, u_K)),$$

3. when $(v_M, v_N) \xrightarrow{a!} (u_M, u_N)$, $v_K \xrightarrow{a?} u_K$ and $\alpha = a!$, in this case, $a!$, $a?$ are shared interfaces between $M$ and $N$, by the definition of $M \parallel N$,

(a) if $v_M \xrightarrow{a!} u_M$, $v_N = u_N$, in this case, $a?$ is not a shared interface between $N$ and $K$, by the definition of $N \parallel K$, we have $(v_N, v_K) \xrightarrow{a?} (u_N, u_K)$; $a!$, $a?$ are shared interfaces between $M$ and $N \parallel K$, by the definition of $M \parallel (N \parallel K)$, we have

$$(v_M, (v_N, v_K)) \xrightarrow{\alpha} (u_M, (u_N, u_K)),$$
(b) if \( v_M = u_M, v_N \xrightarrow{a_l} u_N, \) in this case, \( a!, a? \) are shared interfaces between \( N \) and \( K \), by the definition of \( N \parallel K \), we have \((v_N,v_K) \xrightarrow{a_l} (u_N,u_K)\); \( a!, a? \) are not shared interfaces between \( M \) and \( N \parallel K \), by the definition of \( M \parallel (N \parallel K)\), by the definition of \( M \parallel (N \parallel K)\), we have

\[
(v_M,(v_N,v_K)) \xrightarrow{\alpha} (u_M,(u_N,u_K)),
\]

4. when \((v_M,v_N) \xrightarrow{a_l} (u_M,u_N), v_K \xrightarrow{a_l} u_K \) and \( \alpha = a_\gamma \), in this case, by the definition of \( M \parallel N \),

(a) if \( v_M \xrightarrow{a_l} u_M, v_N = u_N, \) in this case, \( a! \) is a shared interface between \( N \) and \( K \), by the definition of \( N \parallel K \), we have \((v_N,v_K) \xrightarrow{a_l} (u_N,u_K)\); \( a!, a? \) are shared interfaces between \( M \) and \( N \parallel K \), by the definition of \( M \parallel (N \parallel K)\), we have

\[
(v_M,(v_N,v_K)) \xrightarrow{\alpha} (u_M,(u_N,u_K)),
\]

(b) if \( v_M = u_M, v_N \xrightarrow{a_l} u_N, \) in this case, \( a!, a? \) are shared interfaces between \( N \) and \( K \), by the definition of \( N \parallel K \), we have \((v_N,v_K) \xrightarrow{a_l} (u_N,u_K)\); \( a!, a? \) are not shared interfaces between \( M \) and \( N \parallel K \), by the definition of \( M \parallel (N \parallel K)\), we have

\[
(v_M,(v_N,v_K)) \xrightarrow{\alpha} (u_M,(u_N,u_K)),
\]

So that, we get \((v_M,(v_N,v_K)) \xrightarrow{\alpha} (u_M,(u_N,u_K))\) existing in \( M \parallel (N \parallel K) \).

Vice versa, let

\[
(v_M,(v_N,v_K)) \xrightarrow{\alpha} (u_M,(u_N,u_K))
\]

be a transition in \( M \parallel (N \parallel K) \), we are able to obtain \((v_M,v_N) \xrightarrow{\alpha_\gamma} (u_M,u_N)\) in \( (M \parallel N) \parallel K \).

Obviously, we are able to see \(((u_M,u_N),(u_M,(u_N,u_K))) \in S_2\). Therefore, according to Definition 3.4.4, the binary relation \( S_2 \) is a strong bisimulation. So that,
we prove

\[(M \parallel N) \parallel K = M \parallel (N \parallel K).\]

Finally, by the commutativity we proved above, we have

\[(M \parallel K) \parallel N = M \parallel (K \parallel N) = M \parallel (N \parallel K).\]

Therefore, we get the proof. 

\[\square\]
Chapter 4

Generalising Interface Automata

Interface automata are a class of models which are able to describe the behaviours of participants in synchronous communications. This combined with the fact that interfaces are “participants’-oblivious” makes interface automata not suitable to describe the asynchronous behaviours of local views of choreographies. We propose group interface automata (GLA for short), a generalisation of interface automata.

4.1 Motivations

An output and its corresponding input between two interface automata must be synchronized immediately when composing the automata. In other words, in order to interact two interface automata have to be in states offering complementary communication transitions. Accordingly, we consider “erroneous” a state of the composition of the automata where an output does not have a corresponding input (cf. Definition 3.4.6 on page 43). We illustrate this with the following example. Consider the three interface automata $M$, $K$, and $N$ in Fig. 4.1 where also the products $M \otimes K$ and $(M \otimes K) \otimes N$ are shown. According to Definition 3.4.6, the initial state $v_0w_0u_0$ of $M \otimes N \otimes K$ is an error state (denoted by the red box) since the initial state $u_0$ in $N$ has a shared output $b!$, but the initial state $v_0w_0$ in $M \otimes K$ does not have a corresponding shared input $b?$ to immediately consume $b!$. Hence, the synchronous composition of $M$, $K$ and $N$ is
4.2 Group interface automata

We introduce group interface automata (or GIA for short) which are the extension of interface automata based on an asynchronous channel-based semantics. Group interface automata asynchrony is realised by means of *buffered* channels, namely channels that allow messages from the sender to be stored and consumed later by the receiver. The composition of GIA assumes that each pair of automata $M$ and $N$ uses a channel between $M$ and $N$. For our purposes it is enough to assume that the size of the buffers of each channel is one. And we consider that one-size buffer channels are convenient for the extension of the concept of error states from interface automata to GIA. As

![Fig. 4.1 Error states in the product](image)

...
describe below, our definition of composition induces one-size buffer channels between each two participants. When an automaton wants to execute on output, the message is dispatched to the buffer connected between sender and receiver provided the buffer is empty. Dually, a machine who wants to receive a message access its buffers first, then consumes the messages if there is any.

**Definition 4.2.1 (Group interface automaton).** Let $\mathcal{P}$ be a set of participants, $\mathcal{M}$ be a set of messages. A group interface automaton (GIA for short) $I = (V,v_0,\mathcal{G},A,T)$ is a 5-tuple defined as follows,

1. $V$ is a finite set of states,
2. $v_0 \in V$ is the initial state,
3. $\mathcal{G}$ is the group, where $\mathcal{G} \subseteq \mathcal{P}$,
4. $A = A^I \cup A^O \cup A^H$ is a set of interfaces (actions), where
   \[ A^I = (\mathcal{P} \setminus \mathcal{G}) \times \mathcal{G} \times \{?\} \times \mathcal{M}, \quad A^O = \mathcal{G} \times (\mathcal{P} \setminus \mathcal{G}) \times \{!\} \times \mathcal{M}, \quad A^H = \mathcal{G} \times \mathcal{G} \times \{!?!\} \times \mathcal{M}, \]
5. $T \subseteq V \times A \cup \{\tau\} \times V$ is a set of transitions.

As a matter of fact, $\tau$ is a special internal actions. The usage of them will be explained in Chapter 5. Additionally, we write $v \xrightarrow{\alpha} v'$ instead of $(v,\alpha,v') \in T$, when the set of transitions $T$ is clear from the context. And we shorten interface $(A,B,?,m) \in A^I$, $(A,B,!,m) \in A^O$, $(A,B,!,?,m) \in A^H$ respectively with $AB?m$, $AB!m$, $AB!?!m$. Given a GIA $I = (V,v_0,\mathcal{G},A,T)$, we can write $I_G$ to explicitly highlight the group of $I$ writing $I_A$ when $\mathcal{G}$ is the singleton $\{A\}$.

From Definition 4.2.1, we are aware that a GIA consists of a set of finite states; an initial state; a set of participants; a set of interfaces including input actions, output actions and internal actions and a set of transitions. For instance, Fig. 4.2 shows the visual notation of a GIA $I_B = (V,v_0,\{B\},A,T)$ where

1. $V = \{v_0,v_1,v_2,v_3,v_4\}$
4.3 Composing GIAs

We have to adapt the notion of composability to GIA.

**Definition 4.3.1** (Composability of GIA). Two GIA $I_G$ and $I_{G'}$ are composable if $G \cap G' = \emptyset$.

![Graph](image-url)
When two GIA are composable, we are able to find the shared interfaces (actions) between these two GIA in advance.

**Definition 4.3.2 (Shared interface).** Given two composable GIA $\mathcal{I} = (V, v_0, \mathcal{G}, \mathcal{A}, \mathcal{T})$ and $\mathcal{I}' = (V', v'_0, \mathcal{G}', \mathcal{A}', \mathcal{T}')$, we define the shared inputs, shared outputs and shared internals between $\mathcal{I}$ and $\mathcal{I}'$ are $\text{si}(\mathcal{I}, \mathcal{I}')$, $\text{so}(\mathcal{I}, \mathcal{I}')$, $\text{sh}(\mathcal{I}, \mathcal{I}')$ respectively, where

\[
\begin{align*}
\text{si}(\mathcal{I}, \mathcal{I}') &= \{ AB?m \in \mathcal{A} \mid A \in \mathcal{G}' \} \cup \{ AB?m \in \mathcal{A}' \mid A \in \mathcal{G} \}, \\
\text{so}(\mathcal{I}, \mathcal{I}') &= \{ AB!m \in \mathcal{A} \mid B \in \mathcal{G}' \} \cup \{ AB!m \in \mathcal{A}' \mid B \in \mathcal{G} \}, \\
\text{sh}(\mathcal{I}, \mathcal{I}') &= \{ AB!?m \mid (AB!m \in \mathcal{A} \land AB?m \in \mathcal{A}') \text{ or } (AB!m \in \mathcal{A}' \land AB?m \in \mathcal{A}) \}.
\end{align*}
\]

The interactions between two composable GIA are formalised in the following definition which highlights the role of interfaces of GIA.

**Definition 4.3.3 ($\otimes$-product of two GIA).** Let $\mathcal{I}' = (V', v'_0, \mathcal{G}', \mathcal{A}', \mathcal{T}')$ and $\mathcal{I}'' = (V'', v''_0, \mathcal{G}'', \mathcal{A}'', \mathcal{T}'')$ be two composable GIA, $\mathcal{I}' \otimes \mathcal{I}''$ is a GIA $\mathcal{I} = (V, v_0, \mathcal{G}, \mathcal{A}, \mathcal{T})$ defined as follows.

\[
\begin{align*}
V &= V' \times V'', \\
v_0 &= (v'_0, v''_0), \\
\mathcal{G} &= \mathcal{G}' \cup \mathcal{G}'', \\
\mathcal{A} &= \mathcal{A}^I \cup \mathcal{A}^O \cup \mathcal{A}^H,
\end{align*}
\]

where the set of receive actions $\mathcal{A}^I$, send actions $\mathcal{A}^O$ and internal actions $\mathcal{A}^H$ of $\mathcal{I}' \otimes \mathcal{I}''$ are defined by

\[
\begin{align*}
\mathcal{A}^I &= (\mathcal{A}'^I \cup \mathcal{A}''^I) \setminus \text{si}(\mathcal{I}', \mathcal{I}''), \\
\mathcal{A}^O &= (\mathcal{A}'^O \cup \mathcal{A}''^O) \setminus \text{so}(\mathcal{I}', \mathcal{I}''), \\
\mathcal{A}^H &= \mathcal{A}'^H \cup \mathcal{A}''^H \cup \text{sh}(\mathcal{I}', \mathcal{I}'').
\end{align*}
\]
The set $\mathcal{T}$ is defined by

\[
\mathcal{T} = \{(v', v'') \xrightarrow{\alpha} (u', u'') \mid (v' \xrightarrow{\alpha} u') \in \mathcal{T}' \land \alpha \notin si(\mathcal{I}', \mathcal{I}'') \cup so(\mathcal{I}', \mathcal{I}'') \land v'' \in V'' \}
\]

\[
\cup \{(v', v'') \xrightarrow{\alpha} (u', u'') \mid (v'' \xrightarrow{\alpha} u'') \in \mathcal{T}'' \land \alpha \notin si(\mathcal{I}', \mathcal{I}'') \cup so(\mathcal{I}', \mathcal{I}'') \land v' \in V' \}
\]

\[
\cup \{(v', v'') \xrightarrow{AB!m} (u', u'') \mid (v' \xrightarrow{AB!m} u') \in \mathcal{T}' \land (v'' \xrightarrow{AB!m} u'') \in \mathcal{T}'' \}
\]

\[
\cup \{(v', v'') \xrightarrow{BA?m} (u', u'') \mid (v'' \xrightarrow{BA?m} u'') \in \mathcal{T}'' \land (v' \xrightarrow{BA?m} u') \in \mathcal{T}' \}.
\]

The binary operation of $\_ \otimes \_$ is commutative and associative. Similar with interface automata, the commutativity and associativity of the binary operation $\_ \otimes \_$ between GIA are proved by bisimulations. Following [43, 46], we define weak and strong bisimulations.

Before defining the weak bisimulation, we introduce the following notations.

- $v(\tilde{\tau})^* u$ iff $v = v_0 \tilde{\tau} v_1 \tilde{\tau} \cdots \tilde{\tau} v_n = u$

- $v \overset{\tilde{\omega}}{\xrightarrow{\alpha}} u$ iff $v = v_0 (\tilde{\tau})^* \xrightarrow{\alpha_1} (\tilde{\tau})^* v_1 (\tilde{\tau})^* \xrightarrow{\alpha_2} (\tilde{\tau})^* \cdots (\tilde{\tau})^* \xrightarrow{\alpha_n} (\tilde{\tau})^* v_n = u$, where $\tilde{\omega} = \alpha_1 \alpha_2 \cdots \alpha_n$

- $v \overset{\tilde{\omega}}{\xrightarrow{\alpha}} u$ and $\tilde{\omega}$ is obtained from $\omega$ by removing all the $\tau$’s

**Definition 4.3.4** (Weak bisimulation between two GIA). Given two GIA $\mathcal{I}$ and $\mathcal{I}'$, a binary relation $\mathcal{S} \subseteq V \times V'$ is a weak bisimulation if $(v, v') \in \mathcal{S}$ implies, for all $\alpha \in \mathcal{A} \cup \mathcal{A}' \cup \{\tau\}$,

1. if $v \xrightarrow{\alpha} u \in \mathcal{T}$, then, $\exists u' : v' \overset{\tilde{\omega}}{\xrightarrow{\alpha}} u' \land (u, u') \in \mathcal{S}$;

2. if $v' \xrightarrow{\alpha} u' \in \mathcal{T}'$, then, $\exists u : v \overset{\tilde{\omega}}{\xrightarrow{\alpha}} u \land (u, u') \in \mathcal{S}$.

**Definition 4.3.5** (Strong bisimulation between two GIA). Given two GIA $\mathcal{I}$ and $\mathcal{I}'$, a binary relation $\mathcal{S} \subseteq V \times V'$ is a strong bisimulation if $(v, v') \in \mathcal{S}$ implies, for all $\alpha \in \mathcal{A} \cup \mathcal{A}' \cup \{\tau\}$,

1. if $v \xrightarrow{\alpha} u \in \mathcal{T}$, then, $\exists u' : v' \xrightarrow{\alpha} u' \in \mathcal{T}' \land (u, u') \in \mathcal{S}$;
2. If \( v' \xrightarrow{\alpha} u' \in T' \), then, \( \exists u : v \xrightarrow{\alpha} u \in T \wedge (u, u') \in S \).

We write \( I = I' \) if there is a strong bisimulation between two GIA \( I \) and \( I' \). The theorem of the commutativity and associativity of \( \otimes \) operation is shown below and we skip the proof here since the proof of Theorem 4.3.6 is similar with the proof of Theorem 4.5.4.

**Theorem 4.3.6** (Commutativity and associativity of \( \otimes \)-product). Let \( I, I' \) and \( I'' \) be pairwise (mutually) composable GIA. Then, we have

\[
I \otimes I' = I' \otimes I,
\]

\[
(I \otimes I'') \otimes I' = I \otimes (I' \otimes I'').
\]

Fig. 4.3 shows an instance of the \( \otimes \)-product among three mutually composable GIA \( I_A, I_B \) and \( I_C \) since their groups are mutually disjoint (\( \{A\} \cap \{B\} \cap \{C\} = \emptyset \)). Then, we can see \( I_A \) has a shared output \( AC!m \) where its corresponding shared input \( AC?m \) is in \( I_C \), while \( I_B \) has a shared output \( BC!n \) where its corresponding shared input \( BC?n \) is in \( I_C \) as well. Let us consider the \( \otimes \)-product \( I_A \otimes I_C \) first, action \( AC?m \) can synchronize with \( AC!m \) since the shared output \( AC!m \) from \( I_A \) and the shared input \( AC?m \) from \( I_C \). Therefore, in the \( \otimes \)-product \( I_A \otimes I_C \), the shared output \( AC!m \) and the shared input \( AC?m \) are synchronized generating the internal interface \( AC!?m \) of \( I_A \otimes I_C \). Whilst the input \( BC?n \) in \( I_C \) does not have the corresponding output \( BC!n \) in \( I_A \); the input \( BC?n \) is not a shared input between \( I_A \) and \( I_C \). Then, the input \( BC?n \) executes independently without any synchronization and it is retained in \( I_A \otimes I_C \).

Next, let us consider the \( \otimes \)-product \( (I_A \otimes I_C) \otimes I_B \). The internal action \( AC!?m \) in \( I_A \otimes I_C \) executes independently without any synchronizations and retains in \( (I_A \otimes I_C) \otimes I_B \). The GIA \( I_B \) has a shared output \( BC!n \) with a corresponding shared input \( BC?n \) in \( I_A \otimes I_C \). Hence \( BC?n \) synchronizes with \( BC!n \); this yields an internal interface \( BC!?n \) in \( (I_A \otimes I_C) \otimes I_B \). Due to the \( \otimes \)-product operation being commutative and associative,
the orders of computing will not effect the outcome of $\otimes$-product among these three GIA. Therefore, for instance, $(I_A \otimes I_C) \otimes I_B = I_A \otimes (I_B \otimes I_C) = I_A \otimes I_B \otimes I_C$.

Fig. 4.3 $\otimes$-product among three composable GIA

Then, we introduce deadlock states in the $\otimes$-product among GIA, we define

**Definition 4.3.7** (Deadlock states in the $\otimes$-product of GIA). Given two composable GIA $I = (V,v_0,\mathcal{G},A,T)$ and $I' = (V',v'_0,\mathcal{G}',A',T')$, we define a state $(v,v') \in V \times V'$ is a deadlock if $(v,v')$ has only shared inputs actions between $I$ and $I'$ that can not be synchronized.

Fig. 4.4 shows an instance of the $\otimes$-product among three mutually composable GIA $I_A$, $I_B$ and $I_C$ with deadlock states. According to Definition 4.3.7, we can see the states $v_1u_1w_0$ and $v_1u_0w_1$ are both deadlock states in $I_A \otimes I_B \otimes I_C$. The reason is that the states $v_1u_1w_0$, $v_1u_0w_1$ have only shared input transition actions $AB?m$, $AC?n$ respectively, at these two states, the GIA $I_B$ and $I_C$ wait for receiving message $m$, $n$ forever, and do not terminate.
4.3 Composing GIAs

Next, let us see what are unspecified receptions, generally speaking, an unspecified reception state identifies an input action outgoing from the state never receive expected messages. Then, we define

**Definition 4.3.8 (Unspecified reception states in the \(\otimes\)-product of GIA).** Given two composable GIA \(\mathcal{I} = (V, v_0, G, A, \mathcal{T})\) and \(\mathcal{I}' = (V', v_0', G', A', \mathcal{T}')\), we define a state \((v, v') \in V \times V'\) is an unspecified reception if one of the following conditions holds,

- \(\exists v \xrightarrow{AB?m} u \land \forall v' \xrightarrow{\alpha_0} \cdots \xrightarrow{\alpha_n} u' \implies \alpha_{\hat{h}} \neq AB!m\)
- \(\exists v' \xrightarrow{AB?m} u' \land \forall v \xrightarrow{\alpha_0} \cdots \xrightarrow{\alpha_n} u \implies \alpha_{\hat{h}} \neq AB!m\)

where \(AB?m \in \text{si}(\mathcal{I}, \mathcal{I}')\) and \(0 \leq \hat{h} \leq n\) is the minimal index such that \(\alpha_{\hat{h}}\) being an output via buffer \(AB\).

Fig. 4.5 shows an instance of the \(\otimes\)-product among two composable GIA \(\mathcal{I}_A\) and \(\mathcal{I}_B\) with unspecified reception states. According to Definition 4.3.8, the states \(v_0u_0\) is an unspecified reception state in \(\mathcal{I}_A \otimes \mathcal{I}_B\). The states \(u_0\) in \(\mathcal{I}_B\) has an shared input action \(AB?n\), but there is no such corresponding shared output action \(AB!m\) in the sequences from state \(v_0\) in \(\mathcal{I}_A\). Therefore, at the state \(v_0u_0\), the GIA \(\mathcal{I}_B\) waits for receiving message \(n\) forever and does not terminate.
In addition, Fig. 4.5 also shows an orphan message in the $\otimes$-product of composable GIA. We can see that the orphan message $m$ sent from $\mathcal{I}_A$ to $\mathcal{I}_B$ will stay in the buffer $AB$ forever and will not be consumed. Therefore, the state $v_0u_0$ in $\mathcal{I}_A \otimes \mathcal{I}_B$ is not only an unspecified reception state, but also called an error state. We elaborate the notion of error states in the Section 4.4.

### 4.4 Error states

Similar to the error states in the product of two interface automata, the $\otimes$-product of two GIA may generate error states under the one-size semantics, which corresponds to potential orphan messages. Therefore, we define error states in $\otimes$-product of GIA. Before giving the definition, we introduce a notation of dual actions.

\[
dual(AB!m) = AB?m \quad \dual(AB?m) = AB!m
\]

Accordingly, the dual of a string $\alpha_0\alpha_1\cdots\alpha_n$ is defined elementwise:

\[
dual(\alpha_0\alpha_1\cdots\alpha_n) = \dual(\alpha_0)\dual(\alpha_1)\cdots\dual(\alpha_n)
\]
Also, given a transition \( v \xrightarrow{\alpha} u \), we define
\[
\#(v \xrightarrow{\alpha} u, A, B) = \begin{cases} 
\alpha & \text{if } \alpha \text{ is a send/receive action, } \text{sbj}(\alpha) = A \text{ and } \text{obj}(\alpha) = B \\
\tau & \text{otherwise} 
\end{cases}
\]
where \( \tau \) is the empty string. We extend \( \#(\_, \_, \_ \), \text{ to sequences of transitions } t = v \xrightarrow{\alpha_0} \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} u \text{ in the obvious way:} \)
\[
\#(t, A, B) = \#(\alpha_0, A, B) \#(\alpha_1, A, B) \cdots \#(\alpha_n, A, B)
\]

**Definition 4.4.1 (Error states of the } \otimes \text{-product).** Let \( v \) and \( v' \) be states respectively of GIA \( \mathcal{I} \) and \( \mathcal{I}' \). We say that \( v \) has an unmatched shared output by \( v' \) if there exists \( AB!m \in \text{so}(\mathcal{I}, \mathcal{I}') \) and a sequence of transitions \( t = v \xrightarrow{AB!m} \xrightarrow{\beta_0} \cdots \xrightarrow{\beta_k} u \in \mathcal{I} \) then for all \( t' = v' \xrightarrow{\alpha_0} \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} u' \in \mathcal{I}' \) either of the following conditions holds

1. \( \alpha_i \neq AB?m \text{ for all } 0 \leq i \leq n \)

2. there is \( 0 < i \leq n \) such that \( \alpha_i = AB?m \); in this case, let \( \hat{h} \) be the minimal index such that \( \alpha_{\hat{h}} = AB?m \), then there is \( 0 \leq j < \hat{h} \) for which \( \alpha_j \in \text{si}(\mathcal{I}, \mathcal{I}') \cup \text{so}(\mathcal{I}, \mathcal{I}') \) with \( \text{sbj}(\alpha_j) = B \) and
   \[
   \text{obj}(\alpha_j) = C \neq A \implies \text{ro}(\#(t'_{<\hat{h}}, B, C)) \not\in \text{Pref}\left(\text{dual}(\#(t, C, B))\right)
   \]

where \( \text{Pref}(\omega) = \{ \mu | \exists \nu. \omega = \mu \nu \} \) and \( \text{ro}(\_) \) is defined as
\[
\text{ro}(\omega_\alpha) = \begin{cases} 
\omega & \text{if } \alpha \text{ is an output} \\
\omega_\alpha & \text{otherwise} 
\end{cases}
\]

A state \((v, v')\) of the } \otimes \text{-product of two GIA } \mathcal{I} \text{ and } \mathcal{I}' \text{ is an error state of } \mathcal{I} \otimes \mathcal{I}' \text{ if } v \text{ has unmatched shared output by } v' \text{ or } v' \text{ has an unmatched shared outputs by } v. \text{ We denote the set of error states of } \mathcal{I} \otimes \mathcal{I}' \text{ with } \text{Error}(\mathcal{I}, \mathcal{I}').
4.4 Error states

Actually, to have an error state, a shared output of one GIA is not eventually consumed by the intended recipient. Therefore, our definition of error states is intended to detect the situation where at least one shared output is not consumed. Fig. 4.6 shows three classes of error states represented by red boxes. (1) The state $v_1u_1$ of $I_A \otimes I_B$ is an error state (by (1) in Definition 4.4.1) since the shared output $AB!n$ from state $v_1$ does not have its corresponding input $AB?n$ from state $u_1$. In other words, when $I_A$ wants to send $n$ to $I_B$ at state $v_1$, $I_B$ will never receive $n$ at state $u_1$. (2) The initial state $v_0u'_0$ of $I_A \otimes I'_B$ is an error state (by (2) in Definition 4.4.1): The shared output $AB!m$ departing from $v_0$ in $I_A$ has a corresponding input $AB?m$ in the sequence from $u'_0$ in $I'_B$, however, the first transition from $u'_0$ is $AB?n$ which is another shared interface between $I_A$ and $I'_B$. In other words, $I_A$ wants to send $m$ to $I'_B$ at state $v_0$, but $I'_B$ will never receive $m$ from the state $u'_0$ since $m$ is not the expected message received at state $u'_0$, and $m$ blocks the buffer away from $n$. (3)
The state $v_1u_1$ of $\mathcal{I}_{\{A,B\}} \otimes \mathcal{I}_C$ is an error state (by (2) in Definition 4.4.1). The shared output $BC!y$ from state $v_1$ has the corresponding input $BC?y$ from state $u_1$, and the first transition from $u_1$ is $AC?x$, which is another shared interface between $\mathcal{I}_{\{A,B\}}$ and $\mathcal{I}_C$. However, the corresponding shared output $AC!x$ is not in each sequence from state $v_2$ in $\mathcal{I}_{\{A,B\}}$. In other words, $\mathcal{I}_{\{A,B\}}$ sends $y$ to $\mathcal{I}_C$ at state $v_1$, then sends $m$ to $\mathcal{I}_C$ at state $v_2$, but $y$ is not consumed by $\mathcal{I}_C$ since $\mathcal{I}_C$ wants to first receive $x$ from $\mathcal{I}_{\{A,B\}}$ at state $u_1$, however, $x$ is not sent at state $v_2$ in $\mathcal{I}_{\{A,B\}}$.

**Theorem 4.4.2.** An error state $(v,v')$ exists in $\mathcal{I} \otimes \mathcal{I}'$ if and only if at least one send in the shared output departing from $v$ or $v'$ is never consumed.

**Proof.** In order to prove the theorem, we divide it into two implications.

1. \((\Rightarrow)\) If state $(v,v')$ is an error state in $\mathcal{I} \otimes \mathcal{I}'$, then at least a shared output departs from $v$ or $v'$ has not been consumed.

   We assume that all shared outputs departing from $v$ or $v'$ has been consumed. According to Definition 4.4.1, there is a contradiction that state $(v,v')$ is an error state, therefore, we get the proof.

2. \((\Leftarrow)\) If at least a shared output departs from $v$ or $v'$ has not been consumed, then state $(v,v')$ is an error state in $\mathcal{I} \otimes \mathcal{I}'$.

   We assume that all the buffers are empty at initial and a shared output $AB!m$ sent from state $v$ is not consumed at $(v,v')$. In other words, message $m$ blocks the buffer $AB$ forever. Then, we divide the following cases for state $v'$.

   (a) For a transition $v \xrightarrow{AB!m} u$ in $\mathcal{I}$, the message $m$ is pushed to the buffer $AB$, and $m$ will not be consumed forever. We can obtain that for any successors states of $v'$, there are no $AB?m$ as transition actions. So that, $m$ will never be consumed. Therefore, we get $(v,v')$ is an error state by (1) in Definition 4.4.1.

   (b) For a transition $v \xrightarrow{ABm} u$ in $\mathcal{I}$, the message $m$ is pushed to the buffer $AB$, and $m$ will not be consumed forever, we can obtain that for all traces
4.5 Compatibility

Error states in the $\otimes$-product of GIA correspond to potential orphan messages in communications. In order to avoid error states from the $\otimes$-product of GIA, similarly to what done for interface automata, we introduce $\parallel$-composition of GIA which helps to eliminate all error states and some predecessors of error states in $\otimes$-product of GIA. Before giving the definition of $\parallel$-composition, we define two classes of states in $\otimes$-product of GIA.

Definition 4.5.1 (Incompatible states and Compatible states). The set of incompatible states of the $\otimes$-product $I = I' \otimes I''$ of two GIA $I'$ and $I''$ is the smallest set named $\text{Incomp}(I', I'')$, such that,
• Error($\mathcal{I}', \mathcal{I}'') \subseteq \text{Incomp}(\mathcal{I}', \mathcal{I}'')$, and

• for all $((v', v'') \xrightarrow{\alpha} (u', u'')) \in \mathcal{T}$, if $(u', u'') \in \text{Incomp}(\mathcal{I}', \mathcal{I}'')$ then $(v', v'') \in \text{Incomp}(\mathcal{I}', \mathcal{I}'')$.

The set of compatible states of the $\otimes$-product $\mathcal{I} = \mathcal{I}' \otimes \mathcal{I}''$ of two GIA $\mathcal{I}'$ and $\mathcal{I}''$ are defined as

$$\text{Comp}(\mathcal{I}', \mathcal{I}'') = (V' \times V'') \setminus \text{Incomp}(\mathcal{I}', \mathcal{I}'').$$

**Definition 4.5.2** (Compatibility of two GIA). Two GIA $\mathcal{I}'$ and $\mathcal{I}''$ are compatible if $(v'_0, v''_0) \notin \text{Incomp}(\mathcal{I}', \mathcal{I}'')$, where $v'_0$ and $v''_0$ are the initial states of $\mathcal{I}'$ and $\mathcal{I}''$ respectively, otherwise $\mathcal{I}'$ and $\mathcal{I}''$ are incompatible.

**Definition 4.5.3** ($\parallel$-composition of two GIA). If two GIA $\mathcal{I}'$ and $\mathcal{I}''$ are compatible, their $\parallel$-composition $\mathcal{I}' \parallel \mathcal{I}''$ is the GIA $\tilde{\mathcal{I}} = (\tilde{V}, \tilde{v}_0, \tilde{G}, \tilde{A}, \tilde{T})$ where,

$$\tilde{V} = \text{Comp}(\mathcal{I}', \mathcal{I}''),$$

$$\tilde{v}_0 = (v'_0, v''_0),$$

$$\tilde{G} = G,$$

$$\tilde{A} = A,$$

$$\tilde{T} = \mathcal{T} \cap (\text{Comp}(\mathcal{I}', \mathcal{I}'') \times \tilde{A} \cup \{\tau\} \times \text{Comp}(\mathcal{I}', \mathcal{I}'')),$$

where $\mathcal{I} = (V, (v'_0, v''_0), G, A, T)$ is the $\otimes$-product of $\mathcal{I}'$ and $\mathcal{I}''$.

Similar with the $\otimes$-product operations of GIA, the $\parallel$-composition operations of GIA are also commutative and associative.

**Theorem 4.5.4** (Commutativity and associativity of $\parallel$-composition of GIA). Let $\mathcal{I}$, $\mathcal{I}'$ and $\mathcal{I}''$ be pairwise composable GIA. Then

$$\mathcal{I} \parallel \mathcal{I}' = \mathcal{I}' \parallel \mathcal{I},$$

$$(\mathcal{I} \parallel \mathcal{I}'') \parallel \mathcal{I}' = \mathcal{I} \parallel (\mathcal{I}' \parallel \mathcal{I}'').$$
4.5 Compatibility

**Proof.** In order to prove commutativity of $\parallel$-composition, we first show

$$\mathcal{I} \parallel \mathcal{I}' = \mathcal{I}' \parallel \mathcal{I}.$$ 

Let $\mathcal{I} = (V, v_0, \mathcal{G}, \mathcal{A}, \mathcal{T})$ and $\mathcal{I}' = (V', v'_0, \mathcal{G}', \mathcal{A}', \mathcal{T}')$ be two pairwise composable GIA. We define a binary relation $S_1 \subseteq V_{\mathcal{I}} \times V_{\mathcal{I}'}$, where $S_1 = \{(v, v'), (v', v) \mid v \in V, v' \in V'\}$.

If $\langle (v, v'), (v', v) \rangle \in S_1$, for all $\alpha \in \mathcal{A}_{\mathcal{I}} \cup \{\tau\}$ ($\mathcal{A}_{\mathcal{I}} = \mathcal{A}_{\mathcal{I}'}$), let

$$(v, v') \xrightarrow{\alpha} (u, u')$$

be a transition in $\mathcal{I} \parallel \mathcal{I}'$. By the definition of $\mathcal{I} \parallel \mathcal{I}'$, we have 4 cases:

1. $v \xrightarrow{\alpha} u$ and $v' = u'$,
2. $v' \xrightarrow{\alpha} u'$ and $v = u$,
3. $v \xrightarrow{AB?m} u$ and $v' \xrightarrow{AB?m} u'$, $\alpha = AB!m$,
4. $v \xrightarrow{BA?m} u$ and $v' \xrightarrow{BA?m} u'$, $\alpha = BA!m$.

In case (1), by the definition of $\mathcal{I}' \parallel \mathcal{I}$, we have $(v', v) \xrightarrow{\alpha} (u', u)$, and similarly in case (2), case (3), case (4).

Vice versa, let

$$(v', v) \xrightarrow{\alpha} (u', u)$$

be a transition in $\mathcal{I}' \parallel \mathcal{I}$, we have $(v, v') \xrightarrow{\alpha} (u, u')$ in $\mathcal{I} \parallel \mathcal{I}'$.

Obviously, we are able to see $\langle (u, u'), (u', u) \rangle \in S_1$. Therefore, the binary relation $S_1$ is a strong bisimulation. So that, we prove

$$\mathcal{I} \parallel \mathcal{I}' = \mathcal{I}' \parallel \mathcal{I}.$$ 

In order to prove associativity of $\parallel$-composition

$$(\mathcal{I} \parallel \mathcal{I}'') \parallel \mathcal{I}' = \mathcal{I} \parallel (\mathcal{I}' \parallel \mathcal{I}''),$$
we first show

\[(\mathcal{I} \parallel \mathcal{I}') \parallel \mathcal{I}'' = \mathcal{I} \parallel (\mathcal{I}' \parallel \mathcal{I}'').\]

Let \(\mathcal{I} = (V, v_0, \mathcal{G}, \mathcal{A}, \mathcal{T})\), \(\mathcal{I}' = (V', v_0', \mathcal{G}', \mathcal{A}', \mathcal{T}')\) and \(\mathcal{I}'' = (V'', v_0'', \mathcal{G}'', \mathcal{A}'', \mathcal{T}'')\) be pairwise composable GIA. We define a binary relation \(\mathcal{S}_2 \subseteq V_{\mathcal{I}(\parallel \mathcal{I}')} \times V_{\mathcal{I}(\parallel \mathcal{I}'')}\), where

\[\mathcal{S}_2 = \{(v, v') \mid v \in V, v' \in V', v'' \in V''\}.

If \(((v, v'), v'') \in \mathcal{S}_2\), for all \(\alpha \in \mathcal{A}(\mathcal{I}(\parallel \mathcal{I}')) \cup \{\tau\}\) \((\mathcal{A}(\mathcal{I}(\parallel \mathcal{I}')) \parallel \mathcal{I}'') = \mathcal{A}_{\mathcal{I}(\parallel \mathcal{I}'')})\), then, I am going to show \(\mathcal{S}_2\) is a strong bisimulation. Let

\[((v, v'), v'') \xrightarrow{\alpha} ((u, u'), u'')\]

be a transition in \((\mathcal{I} \parallel \mathcal{I}') \parallel \mathcal{I}''). By the definition of \((\mathcal{I} \parallel \mathcal{I}') \parallel \mathcal{I}''\), we have 9 cases in total:

1. When \(v, v' \xrightarrow{\alpha} (u, u')\) and \(v'' = u''\),

   (a) if \(v \xrightarrow{\alpha} u\) and \(v' = u'\), in this case, \(\alpha\) is not a shared interface among \(\mathcal{I}, \mathcal{I}'\) and \(\mathcal{I}''\). By the definition of \(\mathcal{I}' \parallel \mathcal{I}''\), we have \((v', v'') = (u', u'')\). Then, by the definition of \(\mathcal{I} \parallel (\mathcal{I}' \parallel \mathcal{I}'')\), we have

   \[ (v, (v', v'')) \xrightarrow{\alpha} (u, (u', u''))\],

   (b) if \(v = u\) and \(v' \xrightarrow{\alpha} u'\), in this case, \(\alpha\) is not a shared interface among \(\mathcal{I}, \mathcal{I}'\) and \(\mathcal{I}''\). By the definition of \(\mathcal{I}' \parallel \mathcal{I}''\), we have \((v', v'') \xrightarrow{\alpha} (u', u'')\). Then, by the definition of \(\mathcal{I} \parallel (\mathcal{I}' \parallel \mathcal{I}'')\), we have

   \[ (v, (v', v'')) \xrightarrow{\alpha} (u, (u', u''))\],

   (c) if \(v \xrightarrow{AB!m} u, v' \xrightarrow{AB?m} u'\) and \(\alpha = AB!?m\), in this case, \(AB?m\) is not a shared interface between \(\mathcal{I}'\) and \(\mathcal{I}'\), by the definition of \(\mathcal{I}' \parallel \mathcal{I}''\), we have \((v', v'') \xrightarrow{AB?m} (u', u''); AB!m, AB?m\) are shared interfaces between \(\mathcal{I}\) and
4.5 Compatibility

\[ \mathcal{I}' \parallel \mathcal{I}'', \text{ by the definition of } \mathcal{I} \parallel (\mathcal{I}' \parallel \mathcal{I}''), \text{ we have} \]

\[(v, (v', v'')) \xrightarrow{\alpha} (u, (u', u'')), \]

(d) if \( v \xrightarrow{BA\text{?}m} u, v' \xrightarrow{BA!m} u' \) and \( \alpha = BA\text{!}m \), in this case, \( BA\text{!}m \) is not a shared interface between \( \mathcal{I}' \) and \( \mathcal{I}'' \), by the definition of \( \mathcal{I}' \parallel \mathcal{I}'' \), we have \( (v', v'') \xrightarrow{BA!m} (u', u'') \); \( BA\text{!}m \), \( BA\text{?}m \) are shared interfaces between \( \mathcal{I} \) and \( \mathcal{I} \parallel \mathcal{I}' \), by the definition of \( \mathcal{I} \parallel (\mathcal{I}' \parallel \mathcal{I}'') \), we have

\[(v, (v', v'')) \xrightarrow{\alpha} (u, (u', u'')), \]

2. when \((v, v') = (u, u') \) and \( v'' \xrightarrow{\alpha} u'' \), in this case, \( \alpha \) is not a shared interface among \( \mathcal{I}, \mathcal{I}' \) and \( \mathcal{I}'' \). By the definition of \( \mathcal{I} \parallel \mathcal{I}' \), we have \( v = u, v' = u' \). Then, by the definition of \( \mathcal{I}' \parallel \mathcal{I}'' \), we have \( (v', v'') \xrightarrow{\alpha} (u', u'') \). After that, by the definition of \( \mathcal{I} \parallel (\mathcal{I}' \parallel \mathcal{I}'') \), we have

\[(v, (v', v'')) \xrightarrow{\alpha} (u, (u', u'')), \]

3. when \((v, v') \xrightarrow{AC\text{!}m} (u, u') \) and \( v'' \xrightarrow{AC\text{?}m} u'' \) and \( \alpha = AC\text{!}\text{?}m \), in this case, \( AC\text{!}m \), \( AC\text{?}m \) are not shared interfaces between \( \mathcal{I} \) and \( \mathcal{I}' \), by the definition of \( \mathcal{I} \parallel \mathcal{I}' \), we have \( v \xrightarrow{AC\text{!}m} u, v' = u' \); \( AC\text{?}m \) is not a shared interface between \( \mathcal{I}' \) and \( \mathcal{I}'' \), by the definition of \( \mathcal{I} \parallel \mathcal{I}'' \), we have \( (v', v'') \xrightarrow{AC\text{?}m} (u', u'') \); \( AC\text{!}m \), \( AC\text{?}m \) are shared interfaces between \( \mathcal{I} \) and \( \mathcal{I}' \parallel \mathcal{I}'' \), by the definition of \( \mathcal{I} \parallel (\mathcal{I}' \parallel \mathcal{I}'') \), we have

\[(v, (v', v'')) \xrightarrow{\alpha} (u, (u', u'')), \]

4. when \((v, v') \xrightarrow{BC\text{!}m} (u, u') \) and \( v'' \xrightarrow{BC\text{?}m} u'' \) and \( \alpha = BC\text{!}\text{?}m \), in this case, \( BC\text{!}m \), \( BC\text{?}m \) are not shared interfaces between \( \mathcal{I} \) and \( \mathcal{I}' \), by the definition of \( \mathcal{I} \parallel \mathcal{I}' \), we have \( v = u, v' \xrightarrow{BC\text{!}m} u' \); \( BC\text{!}m \), \( BC\text{?}m \) are shared interfaces between \( \mathcal{I}' \) and \( \mathcal{I}'' \), by the definition of \( \mathcal{I}' \parallel \mathcal{I}'' \), we have \( (v', v'') \xrightarrow{\alpha} (u', u'') \); \( BC\text{!}m \), \( BC\text{?}m \) are not shared interfaces between \( \mathcal{I} \) and \( \mathcal{I}' \parallel \mathcal{I}'' \), by the definition of \( \mathcal{I} \parallel (\mathcal{I}' \parallel \mathcal{I}''), \)
we have

\[(v,(v',v'')) \xrightarrow{\alpha} (u,(u',u''))\],

5. when \((v,v') \xrightarrow{CA?m} (u,u')\), \(v'' \xrightarrow{CA!m} u''\) and \(\alpha = CA!m\), in this case, \(CA?m\), \(CA!m\) are not shared interfaces between \(I\) and \(I'\), by the definition of \(I \parallel I'\), we have \(v \xrightarrow{CA?m} u\), \(v' = u'\); \(CA!m\) are not shared interfaces between \(I'\) and \(I''\), by the definition of \(I' \parallel I''\), we have \((v',v'') \xrightarrow{CA!m} (u',u'')\); \(CA?m\), \(CA!m\) are shared interfaces between \(I\) and \(I' \parallel I''\), by the definition of \(I \parallel (I' \parallel I'')\), we have

\[(v,(v',v'')) \xrightarrow{\alpha} (u,(u',u''))\].

6. when \((v,v') \xrightarrow{CB?m} (u,u')\), \(v'' \xrightarrow{CB!m} u''\) and \(\alpha = CB!m\), in this case, \(CB?m\), \(CB!m\) are not shared interfaces between \(I\) and \(I'\), by the definition of \(I \parallel I'\), we have \(v = u\), \(v' \xrightarrow{CB?m} u'\); \(BC!m\), \(BC?m\) are shared interfaces between \(I'\) and \(I''\), by the definition of \(I' \parallel I''\), we have \((v',v'') \xrightarrow{\alpha} (u',u'')\); \(BC!m\), \(BC?m\) are not shared interfaces between \(I\) and \(I' \parallel I''\), by the definition of \(I \parallel (I' \parallel I'')\), we have

\[(v,(v',v'')) \xrightarrow{\alpha} (u,(u',u''))\].

So that, we get \((v,(v',v'')) \xrightarrow{\alpha} (u,(u',u''))\) existing in \(I \parallel (I' \parallel I'')\).

Vice versa, let

\[(v,(v',v'')) \xrightarrow{\alpha} (u,(u',u''))\]

be a transition in \(I \parallel (I' \parallel I'')\), we have \((((v,u'),v'') \xrightarrow{\alpha} ((u,u'),u''))\) in \((I \parallel I') \parallel I''\).

Obviously, we are able to see \((((u,u'),u''),(u,(u',u'')))) \in S_2\). Therefore, the binary relation \(S_2\) is a strong bisimulation. So that, we prove

\((I \parallel I') \parallel I'' = I \parallel (I' \parallel I'')\).
Finally, by the commutativity we proved above, we have

\[(I \parallel I') \parallel I'' = I \parallel (I'' \parallel I') = I \parallel (I' \parallel I'').\]

Therefore, we get the proof.
Chapter 5

Verifying G-Choreographies with GIA

As we mentioned earlier, GIA are suitable models for local views of choreographies. In this chapter, we show how they can be used to check a variant of well-formedness of g-choreographies.

5.1 Another semantics of g-choreographies

We extend the semantics given in [24, 51] to encompass iteration. Iterating a pomsets consists in taking the union of all possible finite or infinite sequential of its compositions. Formally,

**Definition 5.1.1** (Pomsets for iterations). Let \( r = [E, \leq, \lambda] \) be a pomset. The set of pomsets for the iteration Iter\( (r) \) of \( r \) is defined as

\[
\text{Iter}(r) = \bigcup_{n > 0} \text{iter}^n(r),
\]

where

\[
\text{iter}^1(r) = r \quad \text{and} \quad \text{iter}^n(r) = \text{seq}(r, \text{iter}^{n-1}(r)).
\]
5.1 Another semantics of g-choreographies

For instance, Fig. 5.1 shows the pomsets for the iteration of an interaction $\ast\{A \xrightarrow{m} B\}@A$. According to Definition 5.1.2 below, the semantics of the iteration is an infinite set of pomsets, where each element corresponds to an unfolding of the body of the iteration.

We have reviewed the pomset semantics and well-formedness of g-choreographies in Section 3.3. As shown well-sequencedness $ws(G, G')$, well-forkedness $wf(G, G')$ and well-branchedness $wb(G, G')$ are three different constraints for defining the semantics of sequential compositions, parallel compositions and branching compositions of g-choreographies. Now, we consider a notion of well-formedness that relaxes well-sequencedness and, unlike [24, 51], assume that sequential composition is always defined, namely that the relation $ws(\_ , \_ )$ is a tautology on well-formed g-choreographies. Basically, a g-choreography $G' ; G''$ is well-sequenced as long as $G'$ and $G''$ are defined. Fig. 5.2 shows the difference between the notion of well-sequencedness seen in Section 3.3

(cf. page 24) and our relaxed notion. In the earlier notion of well-sequencedness, the sequential composition $G = A \xrightarrow{m} B; B \xrightarrow{n} C$ is well-sequenced while the composition $G' = A \xrightarrow{m} B; C \xrightarrow{n} D$ is not because $G'$ cannot be realised unless extra (unspecified) interactions are arbitrarily added. Instead in our relaxed notion the semantics of $G$ and
5.2 GIA-based Verification: Theory

G’ are both defined. Since the semantics of $A \xrightarrow{m} B$ and $C \xrightarrow{n} D$ are defined. Basically, we do not require the existence of causal dependencies between the sequential composition components. We motivate our decision to relax the notion of well-sequencedness using an analogy on assignments. Consider the statement $x := 4; y := 5$. Since the two assignments are independent, they may be executed in any order (for instance, a compiler could swap the assignments in order to apply some code optimisation).

Likewise, $A \xrightarrow{m} B; C \xrightarrow{n} D$ can be thought of as being equivalent to $A \xrightarrow{m} B | C \xrightarrow{n} D$.

To sum up, we adopt a naive well-formedness on g-choreographies that relaxes well-sequencedness (Definition 3.3.2) and just requires well-forkedness (Definition 3.3.3) and well-branchedness (Definition 3.3.4).

**Definition 5.1.2 (A variant semantics of global choreographies).** The semantics of a global choreography is a family of pomsets defined as

$$[0] = \{\epsilon\}$$

$$[[A \xrightarrow{m} B]] = \{[[\{e_1, e_2\}, \{(e_1, e_1), (e_1, e_2), (e_2, e_2)\}], \lambda]\} \text{ where } \lambda = \begin{cases} e_1 \mapsto AB!m \\ e_2 \mapsto AB?m \end{cases}$$

$$[[G; G’]] = \{\text{seq}(r, r') \mid (r, r') \in [[G]] \times [[G’]]\}$$

$$[[G \mid G’]] = \begin{cases} \{\text{par}(r, r') \mid (r, r') \in [[G]] \times [[G’]]\} & \text{if } \text{wf}(G, G’) \\ \bot & \text{otherwise} \end{cases}$$

$$[[G + G’]] = \begin{cases} [[G]] \cup [[G’]] & \text{if } \text{wb}(G, G’) \\ \bot & \text{otherwise} \end{cases}$$

$$[*\{G\}@A] = \{\text{Iter}(r) \mid r \in [[G]]\}$$

5.2 GIA-based Verification: Theory

We advocate GIA for the verification of well-formedness of g-choreographies. According to [24, 51], well-formedness of g-choreographies implies deadlock freedom. Now, we show
that the naive well-formedness of g-choreographies is equivalent to the non-existence of error states, branching error states and parallel error states in the corresponding GIA. Therefore, non-existence of error states, branching error states and parallel error states implies deadlock freedom. Our approach is schematically shown in Fig. 5.3.

![Fig. 5.3 Research approach on naive well-formedness](image)

Roughly speaking, (1) projection yields a set of GIA \( \{ G_{IA} \mid A \in P \} \) corresponding to each local participant involved in the g-choreography; (2) projection introduces \( \tau \)-transitions, that is internal transitions that do not represent communications; some of those \( \tau \)-transitions may lead to spurious error states in the \( \otimes \)-product between GIA; the elimination of spurious \( \tau \)-transitions, called removable \( \tau \)-transitions, reveals actual error states; then we obtain a set of GIA without removable \( \tau \)-transitions \( \{ \overline{G_{IA}} \mid A \in P \} \); (3) we take the \( \otimes \)-product of all the GIA without removable \( \tau \)-transitions to get \( \otimes_{A \in P}(\overline{G_{IA}}) \); (4) we analyse the GIA \( \otimes_{A \in P}(\overline{G_{IA}}) \) yielded by last step to detect the existence of error states, branching error states and parallel error states.

### 5.2.1 Projections of GIA

We define some auxiliary notions before introducing the projection operation.

1. Let \( \{ u/v \} \mathcal{I} \) be the GIA obtained by substituting the state \( v \) with the state \( u \) in the GIA \( \mathcal{I} \).

2. Let \( \mathcal{I} = (V, v_0, G, A, T) \) and \( \mathcal{I}' = (V', v'_0, G', A', T') \) be two GIA. We define \( \mathcal{I} \times \mathcal{I}' = (V \times V', (v_0, v'_0), G \cup G', A \cup A', T'') \), where \( ((v, v'), \alpha, (u, u')) \in T'' \) iff \( ((v, \alpha, u) \in T \) and \( v' = u' \in V' \) or \( (v', \alpha, u') \in T' \) and \( v = u \in V \).
3. Let $\mathcal{I} \otimes n$ be the automaton $(V \times n, (v_0, n), G, A, \mathcal{T} \otimes n)$, where $\mathcal{T} \otimes n = \{(v, n), (u, n) \mid (v, \alpha, u) \in \mathcal{T}\}$.

4. Let $\mathcal{I} \circ \mathcal{I}'$ be the automaton $(V \cup V', v_0, G \cup G', A \cup A', \mathcal{T} \cup \mathcal{T}')$. This operation is intended to help with connecting two automata by sequential or by branching.

Hereafter, we define a projection function by induction on the syntax of $G$ returning a triple $(\mathcal{I}_A, v_0, v_e)$, where $\mathcal{I}_A$ is a GIA, $v_0$ is its initial state, and $v_e$ is the special state of $\mathcal{I}$ used to connect it to the other GIA. Generally, the connecting state is the final state which does not have any leaving transitions. Here, we use $(\mathcal{I}_A, v_0, v_e) \otimes n$ to represent $(\mathcal{I}_A \otimes n, (v_0, n), (v_e, n))$. 
Definition 5.2.1 (Projection from g-choreographies to GIA). Let $G$ be a g-choreography. The projections $G_{↓A}$ are defined as follows.

\[
G_{↓A} = \begin{cases} 
\begin{array}{ll}
\tau & v_0 \\
v_0 & v_e \\
ABm & v_0 \\
BAm & v_0 \\
\end{array} & \text{if } G = 0 \\
\begin{array}{ll}
\tau & v_0 \\
v_0 & v_e \\
\end{array} & \text{if } G = B \xrightarrow{m} C \\
\begin{array}{ll}
ABm \xrightarrow{\text{if } G = A \xrightarrow{m} B, v_0 \neq v_e} & v_e \\
\end{array} & \text{if } G = A \xrightarrow{m} B, v_0 \neq v_e \\
\begin{array}{ll}
BAm \xrightarrow{\text{if } G = B \xrightarrow{m} A, v_0 \neq v_e} & v_e \\
\end{array} & \text{if } G = B \xrightarrow{m} A, v_0 \neq v_e \\
\end{cases}
\]

\[
(I'_A \circ \{v_e/u_0\} I''_A, v_0, u_e)
\]

where in the last two cases we define
We inductively define the projections and preserve all the behaviours of each participant. Roughly speaking, (1) an empty g-choreography 0 induces a GIA with two states connected by a $\tau$ transition, which are the initial state and the connecting state. (2) An interaction $A \xrightarrow{m} B$ yields three different projections on the sender, receiver and other participants: Each projection has an initial state, a connecting state and a transition, where the transition action is an output (resp. input, $\tau$) if the projection is on the sender (resp. receiver, other participants). (3) The projection $(G'; G'')_{\downarrow A}$ is yielded by merging the connecting state of $G'_{\downarrow A}$ with the initial state of $G''_{\downarrow A}$. (4) The projection $(G' + G'')_{\downarrow A}$ is constructed by merging the initial states and the connecting states of $G'_{\downarrow A}$ and $G''_{\downarrow A}$. (5) The projection $(G' | G'')_{\downarrow A}$ is generated by interleaving the transitions of $G'_{\downarrow A}$ and $G''_{\downarrow A}$, where $G'_{\downarrow A}$ and $G''_{\downarrow A}$ are refined GIA of remove all removable $\tau$ transitions (we will introduce in the next section).

In order to obtain the projection from iterations, there are four special GIA $I_{ls}$, $I_{es}$, $I_{lr}$, $I_{er}$ to help with construction. The GIA $I_{ls}$ and $I_{es}$ correspond to the participant deciding to continue or exit the iteration respectively. While, the GIA $I_{lr}$ and $I_{er}$ correspond to the participants which do not decide to continue or exit the iteration respectively. In $I_{ls}$ and $I_{es}$, the symbols $B_1, \cdots, B_h$ denote the rest of participants in $G$ except $A$. In fact, the sequence of transitions in $I_{ls}$ and $I_{es}$ can be in any order. Then, we start to roughly explain the projections on iterations. (1) The projection $(*\{G'\} \downarrow A)$ is constructed by merging the connecting state of $I_{ls}$ with the initial state of $G'_{\downarrow A}$, as well as merging the connecting state of $G'_{\downarrow A}$ with the initial state of $I_{ls}$, then, merging the initial state of $I_{es}$ with the connecting state of $G'_{\downarrow A}$. (2) Similarly, the projection $(*\{G'\} \downarrow B)$ is constructed by merging the connecting state
5.2 GIA-based Verification: Theory

Fig. 5.4 Instances of projection from g-choreographies
of $\mathcal{I}_{lr}$ with the initial state of $G'_\downarrow A$, as well as merging the connecting state of $G'_\downarrow A$ with the initial state of $\mathcal{I}_{lr}$, then, merging the initial state of $\mathcal{I}_{er}$ with the connecting state of $G'_\downarrow A$.

Fig. 5.4 shows four instances of obtaining GIA as projections from sequential compositions, parallel compositions, branching compositions and iterations of g-choreographies.

### 5.2.2 Removability of $\tau$-transitions

The projection of interactions introduces $\tau$-transitions, which may in turn induce error states in the $\otimes$-product. However, some error states may be spurious, that is, they do not correspond to orphan messages. For instance, Fig. 5.5 shows a well-formed g-choreography $G$, a non well-formed g-choreography $G'$, and their projections. The states $v_3u_1w_2$ and $v_3u_2w_1$ in $\mathcal{I}_A \otimes \mathcal{I}_B \otimes \mathcal{I}_C$ and $v_0u_1w_1$, $v_0u_0w_1$, $v_0u_1w_0$ in $\mathcal{I}_D \otimes \mathcal{I}_E \otimes \mathcal{I}_F$ are error states according to Definition 4.4.1. However, the error states in $\mathcal{I}_A \otimes \mathcal{I}_B \otimes \mathcal{I}_C$...
are spurious error states since the two transition actions from \( w_1 \) and \( w_2 \) are both receive actions, we consider that the two \( \tau \)-transitions in \( I_C \) are removable. The error states in \( I_D \otimes I_E \otimes I_F \) do reflect deadlocks in \( G' \).

We now define the language between two states in GIA. Let \( \omega \) range over the words or communication actions or \( \tau \) and \( \omega' \) be as usual the concatenation of words \( \omega \) and \( \omega' \) with \( \tau \) the neutral element of concatenation.

**Definition 5.2.2.** Given a GIA \( I = (V, v_0, G, A, T) \), the set of words between two states \( v \in V \) and \( v' \in V \) is defined by

\[
L(v, v') = \begin{cases} 
\{ \alpha_0 \cdots \alpha_n | v \xrightarrow{\alpha_0} v_1 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} v' \} & \text{if } v \neq v' \\
\{ \alpha_0 \cdots \alpha_n | v \xrightarrow{\alpha_0} v_1 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} v' \} \cup \{ \tau \} & \text{if } v = v'.
\end{cases}
\]

Intuitively a \( \tau \)-transition is removable when it does not alter the choices behaviour of the GIA.

**Definition 5.2.3 (Removable \( \tau \)-transitions).** Given a GIA \( I = (V, v_0, G, A, T) \), a \( \tau \)-transition \( v \xrightarrow{\tau} v' \in T \) is removable if, for all \( v' \in V \), \( L(v, v') \neq \emptyset \) and for all \( \alpha \omega \in L(v, v') \) we have

\[
\alpha \omega \neq \tau \implies \exists v'' \in V : L(v', v'') = \{ \tau \} \land v' \alpha \omega \text{-compatible to } v''
\]

where \( v' \) is \( \alpha \omega \)-compatible to \( v'' \) if for all \( v' \in V \) such that \( L(v', v') \neq \emptyset \wedge L(v', v) \neq \emptyset \)

1. if there is a sequence in \( L(v'_r, \bar{v}) \) starting with a communication action different than \( \alpha \) then for all \( \beta \omega' \in L(v'_r, \bar{v}) \) such that \( \beta \neq \alpha \) and either both \( \beta \) and \( \alpha \) are output actions or they are both input actions;

2. otherwise, for all \( \bar{v} \in V \) reachable from \( \bar{v} \) with some communication transitions, then

\[
L(v'_r, \bar{v})L(\bar{v}, \bar{v}) = \{ \alpha \omega \} \quad \text{and} \quad L(\bar{v}, v') \in \{ \emptyset, \{ \tau \} \} \ni L(v', \bar{v})
\]
Roughly speaking, a $\tau$-transition $v \xrightarrow{\tau} v_\tau$ is removable when either the two following conditions hold. (1) The $\tau$-transition $v \xrightarrow{\tau} v_\tau$ does not affect the non-deterministic choices from the state $v$. (2) The first non $\tau$-transition actions leaving from states $v$ and $v_\tau$ are all input interfaces or all output interfaces.

For instance, Fig. 5.6 shows four simple instances of the removable $\tau$-transition in the projection. (1) The $\tau$-transitions $v \xrightarrow{\tau} v_\tau$ and $v \xrightarrow{\tau} v'_\tau$ denoted by red colour in $\mathcal{I}_A$ are removable; the $\tau$-transitions $v \xrightarrow{\tau} v_\tau$ and $v_2 \xrightarrow{\tau} v_f$ denoted by red colour in $\mathcal{I}_B$ are removable as well (by (2) in Definition 5.2.3). (2) When $\alpha, \beta$ are both inputs or outputs, the $\tau$-transitions $v \xrightarrow{\tau} v_\tau$ and $v \xrightarrow{\tau} v'_\tau$ denoted by red colour in $\mathcal{I}_C$ are removable; when $\alpha, \beta$ are both inputs or outputs, the $\tau$-transitions $v \xrightarrow{\tau} v_\tau$ and $v_2 \xrightarrow{\tau} v_\alpha$ denoted by red colour in $\mathcal{I}_D$ are removable (by (1) in Definition 5.2.3).

Moreover, let us consider Fig. 5.7. (1) All $\tau$-transitions are removable in $\mathcal{I}_A$ since all the $\tau$-transitions will not affect the non-deterministic choice of the states, where the $\tau$-transitions depart. (2) The transition $v_2 \xrightarrow{\tau} v_3$ in $\mathcal{I}_B$ is removable, but the removability of $v_0 \xrightarrow{\tau} v_1$ depends on the actions $\alpha$ and $\beta$. If $\alpha$ and $\beta$ are both outputs or inputs then $v_0 \xrightarrow{\tau} v_1$ is removable otherwise it is not. (3) The $\tau$-transitions $v_0 \xrightarrow{\tau} v_1$ in the GIA $\mathcal{I}_C$ is removable if $\alpha$ and $\gamma$ are both outputs or inputs, otherwise, the $\tau$ transition is not removable. (4) The $\tau$ transition $v_0 \xrightarrow{\tau} v_1$ in $\mathcal{I}_D$ is not removable regardless of $\alpha, \gamma$ being inputs or outputs. (5) The $\tau$-transition $v_0 \xrightarrow{\tau} v_1$ in $\mathcal{I}_E$ is removable if $\alpha$ and $\gamma$ are both outputs or inputs, otherwise, this $\tau$-transition is not removable.
After the introduction of removable $\tau$-transitions of projection, we construct the GIA without removable $\tau$-transitions by obtaining the $\tau$-equivalence classes.

**Definition 5.2.4 ($\tau$-Equivalence class of states).** Let $\simeq_\tau$ be the smallest equivalence relation containing all $v \simeq_\tau v'$, where $v \xrightarrow{\tau} \ldots \xrightarrow{\tau} v'$ or $v' \xrightarrow{\tau} \ldots \xrightarrow{\tau} v$, and each $\tau$-transition is removable. We write $[v]_{\simeq_\tau}$ as the equivalence class of the state $v$ w.r.t. $\simeq_\tau$.

**Definition 5.2.5.** Let $\mathcal{I} = (V, v_0, G, A, T)$ be a GIA. Let $\mathcal{I} = (\mathcal{V}, v_0, G, A, T)$ be the GIA without removable $\tau$-transitions constructed from $\mathcal{I}$, such that,

1. $\mathcal{V} = \{[v]_{\simeq_\tau} \mid v \in V\} \cup \{\emptyset\}$,
2. $v_0 = [v_0]_{\simeq_\tau}$,
3. $\mathcal{T} = \{( [v]_{\simeq_\tau}, \alpha, [v']_{\simeq_\tau} ) \mid (v, \alpha, v') \in T\}$.

For instance, Fig. 5.8 shows the GIA with without removable $\tau$-transitions $\mathcal{I}_A$, $\mathcal{I}_B$ and $\mathcal{I}_C$ refined from G in Fig. 5.5, as well as their $\otimes$-product $\mathcal{I}_A \otimes \mathcal{I}_B \otimes \mathcal{I}_C$ which does not have any error states.
For convenient, we write the transition relation $\mathcal{T}$ as

$$\mathcal{T}( [v]_{\sim_\tau}, \alpha ) = \bigcup_{v' \in [v]_{\sim_\tau}} [\mathcal{T}(v', \alpha)]_{\sim_\tau}, \text{ where } \alpha \in \mathcal{A} \cup \{\tau\},$$

then, we define the extended transition relation of GIA.

\begin{align*}
\mathcal{T}^* : & \quad V \times (\mathcal{A} \cup \{\tau\})^* \to 2^V, \\
\text{defined inductively via} & \\
\mathcal{T}^*(v, \tau) =& [v]_{\sim_\tau}, \\
\mathcal{T}^*(v, \omega \alpha) =& \bigcup \{[\mathcal{T}(v', \alpha)]_{\sim_\tau} \mid v' \in \mathcal{T}^*(v, \omega)\} = \bigcup_{v' \in \mathcal{T}^*(v, \omega)} [\mathcal{T}(v', \alpha)]_{\sim_\tau},
\end{align*}

where $v, v' \in V$, $\omega \in \mathcal{A}^*$ and $\alpha \in \mathcal{A} \cup \{\tau\}$.

**Theorem 5.2.7.** Let $G$ be a g-choreography, $\mathcal{G}_{\downarrow A} = \mathcal{I} = (V, v_0, \mathcal{G}, \mathcal{A}, \mathcal{T})$ be a GIA projected from participant $A \in \mathcal{P}$. Let $\overline{\mathcal{G}}_{\downarrow A} = (V, v_0, \mathcal{G}, \overline{\mathcal{A}}, \overline{\mathcal{T}})$ be the GIA without removable $\tau$-transitions constructed from $\mathcal{G}_{\downarrow A}$. Then $\mathcal{G}_{\downarrow A}$ and $\overline{\mathcal{G}}_{\downarrow A}$ are language equivalent.
Proof. We need to show $L(G_{\downarrow A}) = L(G_{\downarrow \overline{A}})$. Note that

$$L(G_{\downarrow A}) = \{ \omega \mid T^*(v_0, \omega) \cap V \neq \emptyset \} \quad \text{and} \quad L(G_{\downarrow \overline{A}}) = \{ \omega \mid T^*(\overline{v_0}, \omega) \cap V \neq \emptyset \}$$

So that, we need to show the extended transition functions of $G_{\downarrow A}$ and $G_{\downarrow \overline{A}}$ are the same, namely that $T^*(v_0, \omega) = T^*(\overline{v_0}, \omega)$. We proceed by induction on the length of the word $\omega$.

**Basic case:** If $|\omega| = 0$, then $\omega = \tau$. So that, we have $T^*(v_0, \tau) = [v_0]_{\simeq_\tau}$, we also have $T^*(\overline{v_0}, \tau) = \overline{v_0} = [v_0]_{\simeq_\tau}$. So, we prove $T^*(v_0, \tau) = T^*(\overline{v_0}, \tau)$.

**Induction:** Assume that

$$T^*(v_0, \mu) = T^*(\overline{v_0}, \mu)$$

and let $\omega = \mu \alpha$, where $\alpha$ is the last transition action of $\omega$. According the definition of extended transitions of GIA, we compute

$$T^*(v_0, \omega) = T^*(v_0, \mu \alpha)$$

$$= \bigcup \{ [T(v', \alpha)]_{\simeq_\tau} \mid v' \in T^*(v_0, \mu) \}$$

$$= \bigcup_{v' \in T^*(v_0, \mu)} [T(v', \alpha)]_{\simeq_\tau}$$

$$= T(T^*(v_0, \mu), \alpha)$$

$$= T(T(\overline{v_0}, \mu), \alpha)$$

$$= T(\overline{v_0}, \mu \alpha)$$

$$= T(\overline{v_0}, \omega).$$

Therefore, we obtain

$$T^*(v_0, \omega) = T^*(\overline{v_0}, \omega).$$

Then, we know $G_{\downarrow A}$ and $G_{\downarrow \overline{A}}$ are language equivalent. \qed
After that, we review the branching bisimulation between GIA. Similar with weak and strong bisimulation, branching bisimulation is also a notion of behavioural equivalence for systems, it is able to generalizing strong bisimulation to ignore internal actions while preserving the branching structures of systems. And branching bisimulation is more restrictive than weak bisimulation. According to [52, 53], we define

**Definition 5.2.8** (Branching bisimulation between two GIA). *Given two GIA \( \mathcal{I} \) and \( \mathcal{I}' \), a binary relation \( \mathcal{S} \subseteq V \times V' \) is a branching bisimulation if it is symmetric and satisfies the following transfer property: if \( (v,v') \in \mathcal{S} \) and \( v \xrightarrow{\alpha} u \), then either \( \alpha = \tau \) and \( (u,v') \in \mathcal{S} \), or \( \exists u',w' \text{ such that } v' \xrightarrow{\tau} u' \xrightarrow{\alpha} w', (v,u') \in \mathcal{S} \) and \( (u,w') \in \mathcal{S} \).*

However, in our case of removing \( \tau \)-transitions in GIA, we consider branching bisimulation is not applicable. For instance, let us consider the two GIA in Fig. 5.9, according to Definition 5.2.8, the GIA \( \mathcal{I} \) and \( \mathcal{I}' \) are branching bisimilar with each other. But we account the \( \tau \)-transition of \( v_0 \xrightarrow{\tau} v_1 \) in \( \mathcal{I} \) is not removable since \( \alpha, \beta \) are input and output actions respectively. So that, \( \mathcal{I}' \) is not the refined GIA of \( \mathcal{I} \).

![Fig. 5.9 Branching bisimulation between GIA](image)

In addition, we consider that weak bisimulation is not applicable for removing \( \tau \)-transition in GIA either. For instance, Fig. 5.10 shows two GIA \( \mathcal{I} \) and \( \mathcal{I}' \), according to Definition 4.3.4, the GIA \( \mathcal{I} \) and \( \mathcal{I}' \) are weak bisimilar. However, the transition of \( v_0 \xrightarrow{\tau} v_1 \) in \( \mathcal{I} \) is not removable since \( \alpha, \beta \) are input and output actions respectively. Therefore, \( \mathcal{I}' \) is not the refined GIA of \( \mathcal{I} \).
5.2 GIA-based Verification: Theory

5.2.3 Branching error state

In GIA-based verification, we need to supplement error states with a new concept called branching error states. As we have seen, error states capture orphan messages in communications. However, the non-existence of error states is not sufficient to guarantee the well-formedness of branching compositions. This leads to the definition of branching error states. Again, we need some auxiliary notation first. Recall the definition of \( \text{sbj} \) and \( \text{obj} \) from Section 3.2. We build the auxiliary notations for transition actions of GIA.

\[
\text{sbj}(\tau) = \text{obj}(\tau) = \text{sbj}(\tau) = \emptyset,
\]

\[
\text{sbj}(AB!!m) = \{A\}, \quad \text{obj}(AB!!m) = \{B\}, \quad \text{sbj}(AB??m) = \{A, B\}
\]

\[
\text{sbj}\{\omega \mid \omega = \alpha_0\alpha_1\cdots\alpha_n\} = \bigcup_{0 < i < n} \text{sbj}(\alpha_i), \quad \text{obj}\{\omega \mid \omega = \alpha_0\alpha_1\cdots\alpha_n\} = \bigcup_{0 < i < n} \text{obj}(\alpha_i),
\]

\[
\text{sbj}\{\omega \mid \omega = \alpha_0\alpha_1\cdots\alpha_n\} = \bigcup_{0 < i < n} \text{sbj}(\alpha_i)
\]

**Definition 5.2.9** (Branching error states). *Given a branching composition \( G = G' + G'' \), states \( v \) and \( v' \) are branching error states in \( \otimes_{A \in P(G' + G'')} \downarrow_A = (V, v_0, G, A, T) \) if either of the following conditions holds*

1. \( |T| = 1 \),

2. \( \exists v, u \neq w \in V : \mathcal{L}(v_0, v) = \{\tau\} \land (\exists (v \xrightarrow{AB!!m} u \land v \xrightarrow{CD??n} w) : A \neq C \lor AB??m = CD??n) \),

Fig. 5.10 Weak bisimulation between GIA
3. \( \exists v \neq v' \in V : (L(v_0, v) \neq \{\tau\} \lor L(v_0, v') \neq \{\tau\}) \land (\exists (v \xrightarrow{\alpha} u \land v' \xrightarrow{\alpha} u'), \alpha \neq \tau : \)
\[
(sbj(\alpha) \cap sbj(L(v_0, v))) = \emptyset \land sbj(\alpha) \cap sbj(L(v_0, v')) = \emptyset) \lor \]
\[
(\emptyset \land \emptyset) = \emptyset).
\]

Let \( brcError(\otimes_{A \in P}(G' + G'')) \) be the set of branching error states of \( \otimes_{A \in P}(G' + G'') \).

As we mentioned above, branching error states are intended to capture the non-well-formedness of branching compositions which can not be detected by error states. With reference to Definition 5.2.9, we now discuss with some examples the different cases yielding branching error states (denoted by a green diamond in the following figures).

- By (1), if the initial state of \( \otimes_{A \in P}(G' + G'') \) has only one transition, then the initial state is a branching error state. For instance, in Fig. 5.11, we consider the initial state \( v_0u_0 \) denoted by a green diamond of \( I_A \otimes I_B \) is a branching error state since the unique transition from \( v_0u_0 \) to \( v_1u_1 \) implies that \( A \) and \( B \)
Fig. 5.13 Verification of the well-formedness of $G_3$
\[ G_4 = A \xrightarrow{m} B; A \xrightarrow{c} C + A \xrightarrow{n} B; A \xrightarrow{c} C \]

Fig. 5.14 Verification of the well-formedness of \( G_4 \)
synchronise only on one of the branches and therefore B is not aware of the other branch.

- By (2), if the initial state of $\otimes A \in \mathcal{P}(G' + G'')_A$ has two transitions with different interactions (different senders or different states reached with the same synchronisation) then some confusion may arise in one of the participants involved about which branch is taken; therefore this situation identifies a branching error state. For instance, in Fig. 5.12 of $I_C \otimes I_D$ is a branching error state since there exist two transitions from $v_0u_0$ to $v_1u_1$, where their transition actions are $CD!?!m$, $DC!?!n$ respectively representing two different senders C and D at $v_0u_0$. Likewise, in Fig. 5.13, the initial state $v_0u_0w_0$ denoted by a green diamond of $I_A \otimes I_B \otimes I_C$ is a branching error state since there are two transitions from $v_0u_0w_0$ to have the same actions $AB!?!m$ and B cannot determine if it has to receive or not the message from C.

- By (3) the initial state of $\otimes A \in \mathcal{P}(G' + G'')_A$ is a branching error state if it can reach two states $v$ and $v'$ having a transition labelled with the same action $\alpha$ performed by a participant not involved in the communications up to $v$ or $v'$.

For instance, in Fig. 5.13, we consider the states $v_1u_1w_0$ and $v_1u_2w_0$ denoted by green diamonds of $I_A \otimes I_B \otimes I_C$ are branching error states since there are two transitions $v_1u_1w_0 \xrightarrow{CB!?!n} v_1u_3w_1$ and $v_1u_2w_0 \xrightarrow{CB!?!n} v_1u_3w_1$, where they have the same transition action and C is not in $\text{subj}(AB!?!m)$. For the same reason in Fig. 5.14, states $v_1u_1w_0$ and $v_2u_1w_0$ of $I_A \otimes I_B \otimes I_C$ are branching error states since there are two transitions $v_1u_1w_0 \xrightarrow{AC!?!x} v_3u_1w_1$ and $v_2u_1w_0 \xrightarrow{AC!?!x} v_3u_1w_1$, where they have the same transition action and C is not in $\text{subj}(AB!?!m)$ and $\text{subj}(AB!?!n)$. 
5.2.4 Parallel error states

In GIA-based verification, besides branching error states, we need to define another type of error-state, called parallel error states since the absence of error states is not sufficient to guarantee the well-formedness of parallel compositions.

**Definition 5.2.10** (Parallel error states). Given a parallel composition $G = G' \mid G''$, state $v$ is a parallel error state in $\bigotimes_{A \in \mathcal{P}} (G' \mid G'')_\downarrow A = (V, v_0, G, A, T)$ if

$$\exists v \xrightarrow{\alpha} u \xrightarrow{\alpha} w \in T, v \xrightarrow{\alpha'} u' \xrightarrow{\alpha'} w \in T : u \neq u' \land \alpha \neq \tau.$$  

Let $\text{parError}(\bigotimes_{A \in \mathcal{P}} (G' \mid G'')_\downarrow A)$ be the set of parallel error states of $\bigotimes_{A \in \mathcal{P}} (G' \mid G'')_\downarrow A$.

As we mentioned above, parallel error states are intended to capture the non well-formedness of parallel compositions which can not be detected by error states. According to Definition 3.3.3, we know that the well-forkedness needs there is no common interactions existing in two threads. And according to Definition 5.2.1, we know the projection of parallel compositions is constructed by interleaving the transitions from two threads. So that, we give Definition 5.2.10 to describe the not well-forkedness. Roughly speaking, if there are four transitions in the corresponding GIA can form a diamond where the four transition actions (not $\tau$) are all same. A vertex of the diamond is a parallel error state. For instance, in Fig. 5.15, the parallel composition $G_5$ is not well-forked. However, we are not able to find any error states in its corresponding GIA $\overline{I}_A \otimes \overline{I}_B$ which is the last ones shown in Fig. 5.15. But we consider the initial state $v_0u_0$ of $\overline{I}_A \otimes \overline{I}_B$ is a parallel error states since there are six diamonds that the four transition actions are $AB!m$.

5.3 Auxiliary lemmas

Before we present our GIA-based verification, we have to define some notations of compositions among GIA. For this we consider pointed GIA, that is triples $(\mathcal{I}, v, v')$
where $I$ is a GIA and $v$ and $v'$ are two states of $I$ used in the composition with other pointed GIA.

**Definition 5.3.1 (Composing GIA).** Let $\hat{I} = (I, v_0, v_e)$, $\hat{I}' = (I', u_0, u_e)$ be two pointed GIA such that $v_0$ and $u_0$ are the initial states of $I$ and $I'$ respectively, $v_e$ and $u_e$ are the connecting states of $I$ and $I'$ respectively. The sequential composition of $\hat{I}$ and $\hat{I}'$ is

$$\hat{I} \circ \hat{I}' = (I \circ \{v_e/u_0\} I', v_0, u_e).$$

Similarly, the branching composition of $\hat{I}$ and $\hat{I}'$ is

$$\hat{I} + \hat{I}' = (\{v_e/u_e\} I \circ \{v_0/u_0\} I', v_0, u_e).$$

We show some auxiliary lemmas, where for readability we use $I_A$, $I'_A$ and $I''_A$ instead of $G_A \downarrow A$, $G'_A \downarrow A$ and $G''_A \downarrow A$ respectively in the following.

**Lemma 5.3.2.** Let $G = G'$; $G''$ and $G$, $G'$, $G''$ be the set of participants of $G$, $G'$, and $G''$ respectively. If there is no causal dependence between the events of $[G']$ and the
events of \( G' \), then

\[
\otimes_{A \in G}(G_A) = \otimes_{A \in G'(G_A)} \otimes \otimes_{A \in G''(G_A)}
\]

Proof. From the hypothesis on \( G \), we have \( G = G' \cup G'' \) and \( G' \cap G'' = \emptyset \). Assume \( I'_A = (I'_A, v'^0_A, v'^e_A), I''_A = (I''_A, v''0_A, v''e_A) \). Then, for all \( A \in G' \),

\[
I_A = I'_A; I''_A = (I'_A \circ \{ v'^e_A / v''0_A \} I''_A, v'^0_A, v'^e_A),
\]

because of \( A \notin G'' \), so that \( I''_A \) just contains one state. Therefore, we are able to obtain \( I_A = I'_A \) since \( I''_A = I''_A \). Similarly, for all \( A \in G'' \), we have \( I_A = I''_A \). Then

\[
\otimes_{A \in G}(G_A) = (\otimes_{A \in G'}(I_A)) \otimes (\otimes_{A \in G''}(I_A))
\]

\[
= (\otimes_{A \in G'}(I'_A)) \otimes (\otimes_{A \in G''}(I''_A))
\]

\[
= I' \otimes I''
\]

This finishes the proof. \( \square \)

Lemma 5.3.3. Given a sequential composition \( G = G'; G'' \), then

\[
I = (\otimes_{A \in (G' \setminus G'')}(I'_A)) \otimes (\otimes_{A \in (G'' \setminus G')}((I''_A))) \otimes (\otimes_{A \in (G' \cap G'')}(I'_A; I''_A)),
\]

where \( I = \otimes_{A \in G}(G_A) \), \( G \) is the set of participants of \( G \), \( G' \) and \( G'' \) are the sets of participants of \( G' \) and \( G'' \) respectively.

Proof. From the given conditions, we have

\[
G = G' \cup G''
\]

Assume \( I'_A = (I'_A, v'^0_A, v'^e_A), I''_A = (I''_A, v''0_A, v''e_A) \). Then, \( \forall A \in (G' \setminus G'') \),

\[
I_A = I'_A; I''_A = (I'_A \circ \{ v'^e_A / v''0_A \} I''_A, v'^0_A, v'^e_A) = I'_A,
\]
similarly, $\forall A \in (G'' \setminus G')$,

$$\mathcal{I}_A = \mathcal{I}'_A ; \mathcal{I}''_A = (\mathcal{I}'_A \circ \{v^{\infty}_A / v^0_A\}) \mathcal{I}''_A, v^0_A, v^{\infty}_A = \mathcal{I}''_A,$$

after that, $\forall A \in (G' \cap G'')$,

$$\mathcal{I}_A = \mathcal{I}'_A ; \mathcal{I}''_A = (\mathcal{I}'_A \circ \{v^{\infty}_A / v^0_A\}) \mathcal{I}''_A, v^0_A, v^{\infty}_e),$$

then,

$$I = (\otimes_{A \in G'\setminus G''}(\mathcal{I}_A)) \otimes (\otimes_{A \in G' \setminus G''}(\mathcal{I}_A)) \otimes (\otimes_{A \in (G' \cap G'')(\mathcal{I}_A)})$$

Therefore, we get the proof. \hfill \Box

**Lemma 5.3.4.** Given a sequential composition $G = G; G''$, if

$$\otimes_{A \in G'}(G_{\downarrow A}) \neq \parallel_{A \in G'}(G''_{\downarrow A}) \quad \text{or} \quad \otimes_{A \in G''}(G''_{\downarrow A}) \neq \parallel_{A \in G''}(G''_{\downarrow A}),$$

then

$$\otimes_{A \in G}(G_{\downarrow A}) \neq \parallel_{A \in G}(G_{\downarrow A}),$$

where $G$, $G'$ and $G''$ are the set of participants of $G$, $G'$ and $G''$ respectively. $G_{\downarrow A}$, $G'_{\downarrow A}$ and $G''_{\downarrow A}$ are the GIA without removable $\tau$-transitions constructed from $G_{\downarrow A}$, $G'_{\downarrow A}$ and $G''_{\downarrow A}$ respectively.

**Proof.** Let

$$I = \otimes_{A \in G}(G_{\downarrow A}),$$

$$I' = \otimes_{A \in G'}(G'_{\downarrow A}) = \otimes_{A \in G}(G_{\downarrow A}),$$

$$I'' = \otimes_{A \in G''}(G''_{\downarrow A}) = \otimes_{A \in G}(G'_{\downarrow A})$$
be the product of projections of \( G, G' \) and \( G'' \) respectively, then, we have

\[
\mathcal{I}' = (\otimes_{A \in (G' \setminus G'')} (\mathcal{I}'_A)) \otimes (\otimes_{A \in (G'' \setminus G'')} (\mathcal{I}'_A)) = (\otimes_{A \in (G' \setminus G'')} (\mathcal{I}'_A)) \otimes (\otimes_{A \in (G'' \setminus G'')} (\mathcal{I}'_A))
\]

\[
\mathcal{I}'' = (\otimes_{A \in (G'' \setminus G'')} (\mathcal{I}'_A)) \otimes (\otimes_{A \in (G'' \setminus G'')} (\mathcal{I}'_A)) = (\otimes_{A \in (G'' \setminus G'')} (\mathcal{I}'_A)) \otimes (\otimes_{A \in (G'' \setminus G'')} (\mathcal{I}'_A))
\]

From Lemma 5.3.3, we have

\[
\mathcal{I} = (\otimes_{A \in (G' \setminus G'')} (\mathcal{I}'_A)) \otimes (\otimes_{A \in (G'' \setminus G'')} (\mathcal{I}'_A)) \otimes (\otimes_{A \in (G'' \setminus G'')} (\mathcal{I}'_A)).
\]

Note that all states of \( \mathcal{I}' \) or \( \mathcal{I}'' \) are also states of \( \mathcal{I} \). Hence, there must be error states in \( \mathcal{I} \) since there exist error states in \( \mathcal{I}' \) or \( \mathcal{I}'' \) by hypothesis. \( \square \)

**Lemma 5.3.5.** Given a parallel composition \( G = G' \parallel G'' \), if \( G \) is well-formed, then, we obtain

\[
\mathcal{I} = \mathcal{I}' \otimes \mathcal{I}'',
\]

where \( \mathcal{I} = \otimes_{A \in \mathcal{G}} (\overline{G}_A) \), \( G \) is the set of participants of \( G \), \( \mathcal{I}' = \otimes_{A \in \mathcal{G}} (\overline{G}_A) \), \( \mathcal{I}'' = \otimes_{A \in \mathcal{G}} (\overline{G}_A) \).

**Proof.** From the given conditions and Definition 5.2.1, we assume that \((v'_{A_1}, \cdots, v'_{A_p}, v''_{A_1}, \cdots, v''_{A_p})\) is the state of \( \mathcal{I}' \otimes \mathcal{I}'' \) and \((v'_{A_1}, \cdots, v'_{A_p}, v''_{A_1}, \cdots, v''_{A_p})\) is the state of \( \mathcal{I} \), where \( v'_{A_1}, \cdots, v'_{A_p} \) are the state of \( \mathcal{I}'_{A_1}, \cdots, \mathcal{I}'_{A_p} \) respectively, \( v''_{A_1}, \cdots, v''_{A_p} \) are the state of \( \mathcal{I}''_{A_1}, \cdots, \mathcal{I}''_{A_p} \) respectively.

Then, we define a binary relation \( S = \{(v'_{A_1}, \cdots, v'_{A_p}, v''_{A_1}, \cdots, v''_{A_p}), (v''_{A_1}, v''_{A_1}, \cdots, v''_{A_p})\}\). Similar to the proof 4.5 for Theorem 4.5.4, we are able to show that \( S \) is a strong bisimulation by induction. Therefore, we get that \( \mathcal{I} = \mathcal{I}' \otimes \mathcal{I}'' \). \( \square \)

**Lemma 5.3.6.** Given two GIA \( I = (V, v_0, A, \mathcal{T}) \) and \( I' = (V', v'_0, A', \mathcal{T}') \), if \( I \) and \( I' \) do not have any synchronized (shared) interfaces, then, we have

\[
I \otimes I' = I \parallel I'.
\]
5.3 Auxiliary lemmas

Proof. We need to show that there is no error state in $\mathcal{I} \otimes \mathcal{I}'$. Assuming that $(v, u)$ is an error state in $\mathcal{I}_A \otimes \mathcal{I}_B$, according to Definition 4.4.1, there is an unmatched shared output departing from $v$ or $u$. Therefore, there is a contradiction with the given condition that $\mathcal{I}$ and $\mathcal{I}'$ do not have any synchronized interfaces. Then, we obtain that there is no error state in $\mathcal{I} \otimes \mathcal{I}'$ as well as $\mathcal{I} \otimes \mathcal{I}' = \mathcal{I} \parallel \mathcal{I}'$.

Lemma 5.3.7. Given a g-choreography $G$, if $G$ is well-formed, then there are neither inputs nor outputs interface in $\otimes_{A \in \mathcal{G}(G \downarrow A)}$.

Proof. In order to get the proof, we divide the given $G$ to two cases as follows.

1. If $G = 0$, then according to Definition 5.2.1, for all $A \in \mathcal{G}$, $\overline{G_{\downarrow A}}$ just has one initial state, while the set of interface is empty. Therefore, there are neither inputs nor outputs interface in $\otimes_{A \in \mathcal{G}(G \downarrow A)}$.

2. If $G \neq 0$, then from Definition 5.2.1, we know that for all inputs or outputs in $\overline{G_{\downarrow A}}$, it must be corresponding outputs or inputs in $\overline{G_{\downarrow B}}$, where $A$ and $B$ do the interaction in $G$. And based on Definition 4.3.3, we know that there are neither inputs nor outputs in $\otimes_{A \in \mathcal{G}(G \downarrow A)}$.

Therefore, we get the proof.

Lemma 5.3.8. Given a branching composition $G = G' + G''$, if $G$ is well-formed, then we have

$$\otimes_{A \in \mathcal{G}(G \downarrow A)} = \otimes_{A \in \mathcal{G}(G' \downarrow A)} + \otimes_{A \in \mathcal{G}(G'' \downarrow A)},$$

where $\mathcal{G}, \mathcal{G}', \mathcal{G}''$ is the set of participants of $G, G', G''$ respectively. $\overline{G_{\downarrow A}}, \overline{G'_{\downarrow A}}$ and $\overline{G''_{\downarrow A}}$ are the GIA without removable $\tau$-transitions constructed from $G_{\downarrow A}$, $G'_{\downarrow A}$ and $G''_{\downarrow A}$ respectively.

Proof. From the given condition and induction hypothesis, we get that $G, G'$ and $G''$ are all well-formed. And there is an active participant and the rest are all passive participants.
Assume that $A$ is the active participant, so that in $G\downarrow A$, the first transition actions from the initial state of $G\downarrow A$ are all different send actions. Similarly, in $G'\downarrow A$ and $G''\downarrow A$, the first transition actions from the initial states of $G'\downarrow A$ and $G''\downarrow A$ are all different send actions.

Then, assume that $B$ is one of passive participants, so that in $G\downarrow B$, the first transition actions from the initial state of $G\downarrow B$ are all different receive actions. Similarly, in $G'\downarrow B$ and $G''\downarrow B$, the first transition actions from the initial states of $G'\downarrow B$ and $G''\downarrow B$ are all different receive actions.

Then, we assume that a state $\vec{v} = (v_A, v_B, \cdots, v_P)$ is the initial state of $\otimes A \in G(\downarrow A)$ and $\otimes A \in G'(\downarrow A) + \otimes A \in G''(\downarrow A)$. When a transition $\vec{v} \xrightarrow{AB?m} \vec{v}'$ in $\otimes A \in G(\downarrow A)$, where $\vec{v}' = (v'_A, v'_B, \cdots, v'_P)$, then, we can obtain that $\vec{v} \xrightarrow{AB?m} \vec{v}'$ is also in $\otimes A \in G'(\downarrow A) + \otimes A \in G''(\downarrow A)$.

Therefore, by the induction, we are able to get that there is a strong bisimulation between $\otimes A \in G(\downarrow A)$ and $\otimes A \in G'(\downarrow A) + \otimes A \in G''(\downarrow A)$. So we get the proof.

\[\square\]

**Definition 5.3.9 (Loop composition between GIA).** The loop composition of GIA $I$ is defined as

\[
*(\hat{I}, \hat{I}_l, \hat{I}_e) = (\{u_e/v_0\}I \circ \{v_e/u_0\}I_l \circ \{v_e/w_0\}I_e, u_e, w_e),
\]

where $\hat{I} = (I,v_0,v_e)$, $\hat{I}_l = (I_l,u_0,u_e)$, $\hat{I}_e = (I_e,w_0,w_e)$ and $v_0$, $u_0$ and $w_0$ are the initial states of $I$, $I_l$ and $I_e$ respectively; $v_e$, $u_e$ and $w_e$ are the special connecting states of $I$, $I_l$ and $I_e$ respectively.

**Lemma 5.3.10.** Given an iteration $G = *\{G'\} \otimes A$, then we obtain

\[
I = *(\hat{I}', \hat{I}_l', \hat{I}_e'),
\]
where $\mathcal{I} = \otimes_{A \in G} (\mathcal{G}_{\downarrow A})$, $\mathcal{I}' = \otimes_{A \in G} (\mathcal{G}_{\downarrow A}^r)$, $\mathcal{G}$ is the set of participants of $\mathcal{G}$ and $\mathcal{G}'$. And $\forall 1 \leq i \leq h : B_i \in \mathcal{G} \setminus \{A\}$.

$\mathcal{I}_l = \begin{array}{c}
\bullet_0 \\
\vdots \\
\bullet_h \\
\end{array}$

Proof. From the given conditions and Definition 5.2.1 and Definition 5.3.9, we are able to obtain that $A \in \mathcal{G}$, assume $\mathcal{I}'_A = (\mathcal{I}'_A, p_0, p_e)$,

$$\mathcal{I}_A = *(\mathcal{I}'_A, \mathcal{I}'_{ls}, \mathcal{I}'_{es}) = (\{u_e/p_0\} \mathcal{I}'_A \circ \{p_e/u_0\} \mathcal{I}_{ls} \circ \{p_e/w_0\} \mathcal{I}_{es}, u_e, w_e).$$

Then, $\forall B \in \mathcal{G} \setminus \{A\}$, assume $\mathcal{I}'_B = (\mathcal{I}'_B, q_0, q_e)$,

$$\mathcal{I}_B = *(\mathcal{I}'_B, \mathcal{I}'_{lr}, \mathcal{I}'_{er}) = (\{u_e/q_0\} \mathcal{I}'_B \circ \{q_e/u_0\} \mathcal{I}_{lr} \circ \{q_e/w_0\} \mathcal{I}_{er}, u_e, w_e).$$

Then, we can also obtain

$$\mathcal{I}' = \mathcal{I}'_A \otimes (\otimes_{B \in \mathcal{G} \setminus \{A\}} \mathcal{I}'_B).$$

Then, we can get the equation as follows.

$$\mathcal{I} = \mathcal{I}_A \otimes (\otimes_{B \in \mathcal{G} \setminus \{A\}} \mathcal{I}_B)$$

$$= *(\mathcal{I}'_A, \mathcal{I}'_{ls}, \mathcal{I}'_{es}) \otimes (\otimes_{B \in \mathcal{G} \setminus \{A\}} \mathcal{I}'_B) \circ (\mathcal{I}'_B, \mathcal{I}'_{lr}, \mathcal{I}'_{er}))$$

$$= *(\mathcal{I}'_A \otimes (\otimes_{B \in \mathcal{G} \setminus \{A\}} \mathcal{I}'_B), \mathcal{I}_l, \mathcal{I}_e)$$

$$= *(\mathcal{I}', \mathcal{I}_l, \mathcal{I}_e)$$

This finishes the proof. $\square$
5.4 GIA-based Well-formedness

The following theorem shows that a g-choreography (Definition 3.1.1) is well-formed (Definition 5.1.2) if, and only if the \( \otimes \)-product (Definition 4.3.3) equals the \( \parallel \)-composition (Definition 4.5.3) and has no branching error states (Definition 5.2.9) and no parallel error states (Definition 5.2.10).

**Theorem 5.4.1.** A g-choreography \( G \) with participants \( G \subseteq P \) is well-formed if and only if

\[
\bigotimes_{A \in G}(G_{\downarrow A}) = \bigparallel_{A \in G}(G_{\downarrow A})
\]

and there are neither branching nor parallel error states in \( \bigotimes_{A \in G}(G_{\downarrow A}) \), and \( G_{\downarrow A} \) is the GIA without removable \( \tau \)-transition constructed from \( G_{\downarrow A} \).

**Proof.** We proceed by induction on the structure of \( G \).

1. If \( G = A \xrightarrow{m} B \) then

\[
[G] = \left[ AB!m \rightarrow AB?m \right]
\]

and the projections of \( G \) are

\[
G_{\downarrow A} = G_{\downarrow A} = \begin{array}{c}
\xrightarrow{AB!m} \\
\xleftarrow{ABm}
\end{array}
\quad G_{\downarrow B} = G_{\downarrow B} = \begin{array}{c}
\xrightarrow{AB?m} \\
\xleftarrow{ABm}
\end{array}
\]

Hence, by definition

\[
G_{\downarrow A} \otimes G_{\downarrow B} = G_{\downarrow A} \parallel G_{\downarrow B}
\]

And there are no branching or parallel error states. This prove the necessity part.

For sufficiency, the proof is straightforward.

2. If \( G = G' ; G'' \)

   (a) \((\Rightarrow)\) Assume \( G \) well-formed, then, \( G' \) and \( G'' \) are well-formed as well. Let

   \( I' = \bigotimes_{A \in G'}(G'_{\downarrow A}) \) and \( I'' = \bigotimes_{A \in G''}(G''_{\downarrow A}) \) be the product of projections of
\(G'\) and \(G''\) respectively, then by the induction hypothesis, we obtain

\[
\mathcal{I}' = \bigotimes_{A \in G'}(\overline{G' \downarrow A}) = \parallel_{A \in G'}(\overline{G' \downarrow A}),
\]
\[
\mathcal{I}'' = \bigotimes_{A \in G''}(\overline{G'' \downarrow A}) = \parallel_{A \in G''}(\overline{G'' \downarrow A}).
\]

In other words, there are no error states in neither \(\mathcal{I}'\) nor \(\mathcal{I}''\). As we know, no error state means that all shared outputs will be consumed in the \(\otimes\)-product. Next, we are going to show there is no error state in \(\mathcal{I}\). From the Lemma 5.3.3, we have

\[
\mathcal{I} = (\bigotimes_{A \in G' \setminus G''}(\mathcal{I}'_A)) \otimes (\bigotimes_{A \in (G'' \setminus G')}((\mathcal{I}''_A)) \otimes (\bigotimes_{A \in (G' \cap G'')}(\mathcal{I}'_A;\mathcal{I}''_A))
\]

By contradiction, assume that there is an error state called \(\vec{v} = (v_A,v_B,\ldots,v_P)\) in \(\mathcal{I}\). According to Definition 4.4.1, there is a shared output \(ABm!\) departing from \(v_A\) and the message \(m\) will never be consumed by the corresponding input \(AB?m\) in all the traces from \(v_B\). Then, there are the following cases

i. If \(A\) and \(B\) are only in \(G'\), then \(v_A,v_B\) are the states of \(\overline{G' \downarrow A}\) and \(\overline{G' \downarrow B}\) respectively. According to the induction hypothesis of no error state existing in \(\mathcal{I}'\), so that there is no such \(AB!m\) from \(v_A\) being unmatched by \(v_B\). In other words, message \(m\) can be received. Therefore, there is a contradiction with the assumption of \(\vec{v}\) being an error state.

ii. If \(A\) is only in \(G'\), \(B\) is in both \(G'\) and \(G''\), Similar to case (2(a)i), there is a contradiction with the assumption of \(\vec{v}\) being an error state.

iii. If \(A\) is in both \(G'\) and \(G''\), \(B\) is only in \(G'\), Similar to case (2(a)i), there is a contradiction with the assumption of \(\vec{v}\) being an error state.

iv. If \(A\) is in both \(G'\) and \(G''\), \(B\) is both in \(G'\) and \(G''\), According to the induction hypothesis of no error state existing in \(\mathcal{I}'\) and \(\mathcal{I}''\), so that there is no such \(AB!m\) from \(v_A\) being unmatched by \(v_B\). In other words,
message $m$ can be received. Therefore, there is a contradiction with the assumption of $\vec{v}$ being an error state.

v. If $A$ is in both $G'$ and $G''$, $B$ is only in $G''$, According to the induction hypothesis of no error state existing in $I''$, so that there is no such $AB!m$ from $v_A$ being unmatched by $v_B$. In other words, message $m$ can be received. Therefore, there is a contradiction with the assumption of $\vec{v}$ being an error state.

vi. If $A$ is only in $G''$, $B$ is in both $G'$ and $G''$, Similar to case (2(a)v), there is a contradiction with the assumption of $\vec{v}$ being an error state.

vii. If $A$ and $B$ are only in $G''$, Similar to case (2(a)v), there is a contradiction with the assumption of $\vec{v}$ being an error state.

Therefore, we prove that the error state $\vec{v} = (v_A, v_B, \cdots, v_P)$ in $I$ does not exist. In other words, there is no error state in $I$, then, we are able to obtain

$$\otimes_{A \in G}(\overline{G_A}) = \parallel_{A \in G}(\overline{G_A}).$$

And according to the Definition 5.2.9 and 5.2.10 and the induction hypothesis, there are no branching, parallel error states in $I'$ and $I''$. Therefore, we are aware of that there are no branching, parallel error states in $\otimes_{A \in G}(\overline{G_A})$.

(b) ($\Leftarrow$) We assume that

$$\otimes_{A \in G}(\overline{G_A}) = \parallel_{A \in G}(\overline{G_A})$$

and no branching, parallel error states exist in $\otimes_{A \in G}(\overline{G_A})$.

Then, we want the follows are both correct.

$$\otimes_{A \in G'}(\overline{G_A}) = \parallel_{A \in G'}(\overline{G_A})$$

$$\otimes_{A \in G''}(\overline{G_A}) = \parallel_{A \in G''}(\overline{G_A})$$
5.4 GIA-based Well-formedness

As a matter of fact, the equations above can also be described as follows.

\[ \otimes_{A \in G} (G'_{\downarrow A}) = \parallel_{A \in G} (G'_{\downarrow A}) \]

\[ \otimes_{A \in G} (G''_{\downarrow A}) = \parallel_{A \in G} (G''_{\downarrow A}) \]

Let \( I = \otimes_{A \in G} (G'_{\downarrow A}) \), \( I' = \otimes_{A \in G} (G'_{\downarrow A}) \) and \( I'' = \otimes_{A \in G} (G''_{\downarrow A}) \) be the product of projections of \( G \), \( G' \) and \( G'' \) respectively, then, we have

\[ I' = (\otimes_{A \in (g' \setminus g'') (I'_{A})}) \otimes (\otimes_{A \in (g'' \setminus g') (I'_{A})}) \otimes (\otimes_{A \in (g' \cap g'') (I'_{A})}), \]

\[ I'' = (\otimes_{A \in (g'' \setminus g'') (I''_{A})}) \otimes (\otimes_{A \in (g' \setminus g'') (I''_{A})}) \otimes (\otimes_{A \in (g' \cap g'') (I''_{A})}). \]

From Lemma 5.3.3, we have

\[ I = (\otimes_{A \in (g' \setminus g'') (I'_{A})}) \otimes (\otimes_{A \in (g'' \setminus g') (I''_{A})}) \otimes (\otimes_{A \in (g' \cap g'') (I'_{A}; I''_{A})}). \]

Therefore, each state and transition in \( I' \) and \( I'' \) is also in \( I \). According to the hypothesis, there are no error states and no branching, parallel error states in \( I \), therefore, we get there are no error states and no branching, parallel error states in \( I' \) and \( I'' \).

Alternatively, according to Lemma 5.3.4, we know its contraposition is correct. Then, we also obtain that there is no error state in \( I' \) and \( I'' \).

Afterwards, according to the induction hypothesis, we obtain that \( G' \) and \( G'' \) are well-formed. Therefore, regardless of existence the causal dependences between the semantic of \( G' \) and the semantic of \( G' \), according to our definition of well-formed g-choreography, \( G = G'; G'' \) is well-formed.

Therefore, we get the proof.

3. If \( G = G' | G'' \)
(a) \(\iff\) We assume that \(G\) is well-formed, then, \(G'\) and \(G''\) are well-formed as well. Let \(I = \bigotimes_{A \in G} (G_{\downarrow A})\), \(I' = \bigotimes_{A \in G'} (G'_{\downarrow A})\) and \(I'' = \bigotimes_{A \in G''} (G''_{\downarrow A})\) be the product of projections of \(G\), \(G'\) and \(G''\) respectively, then by the induction hypothesis, we obtain

\[
I' = \bigotimes_{A \in G'} (G'_{\downarrow A}) = \parallel_{A \in G'} (G'_{\downarrow A});
\]
\[
I'' = \bigotimes_{A \in G''} (G''_{\downarrow A}) = \parallel_{A \in G''} (G''_{\downarrow A}).
\]

In other words, there are no error states, no branching error states and no parallel error states in neither \(I'\) nor \(I''\).

From Lemma 5.3.5, we have

\[I = I' \otimes I''.\]

Therefore, we obtain that \(I\) will not have any error states or branching error states and we are able to obtain

\[
\bigotimes_{A \in G} (G_{\downarrow A}) = \parallel_{A \in G} (G_{\downarrow A}).
\]

Then, we are going to show there is no parallel error state in \(\bigotimes_{A \in G} (G_{\downarrow A})\). According to Definition 5.2.1, we are aware of the each projection of a parallel composition is generated by interleaving the transitions, which belongs to the projections on each thread. Since \(G = G' \parallel G''\) is well-formed, that means there does not exist an interaction both in \(G'\) and \(G''\). Therefore, a diamond which has four edges with same label must not be in each projection. So that, we obtain that parallel error states must not in \(G\).

(b) \(\iff\) We use contra position to help with the proof. So the new statement is that if \(G = G' \parallel G''\) is not a well-formed parallel composition, then, we obtain \(\bigotimes_{A \in G} (G_{\downarrow A}) \neq \parallel_{A \in G} (G_{\downarrow A})\) or there exist branching error states. In other words, if \(G\) is not a well-formed branching, then, there must be error
states or branching, parallel error states in $\otimes_{A \in G}(G_{\downarrow A})$. Let $\mathcal{I} = \otimes_{A \in G}(G_{\downarrow A})$, $\mathcal{I}' = \otimes_{A \in G'}(G'_{\downarrow A})$ and $\mathcal{I}'' = \otimes_{A \in G''}(G''_{\downarrow A})$ be the product of projections of $G$, $G'$ and $G''$ respectively.

So we divide the not well-formed $G = G' \mid G''$ to several cases as follows,

i. If there is at least an interaction appearing in both $G'$ and $G''$, for instance, if the interaction is $A \xrightarrow{m} B$,

the above two diamond must be in $\mathcal{I}_A$ and $\mathcal{I}_B$ respectively. Therefore, according to Definition 4.3.3, 4.4.1, 5.2.10 and Theorem 4.4.2, if there is at least shared output is unmatched in $\mathcal{I}$, we obtain that a error state in $\mathcal{I}$, if all the shared outputs are matched, then, we obtain that a parallel error state in $\mathcal{I}$.

ii. If there is no common interaction appearing both in $G'$ and $G''$.

A. if $G'$ has an error state, that means there is at least an unmatched shared output. Assume $AB!m$ departing from $v_A$ is a shared output unmatched by $v_B$, from the assumption, we know there is no $A \xrightarrow{m} B$ in $G''$, in other words, there is no $AB?m$ in $\mathcal{I}_{B''}$. Then we can get that $AB!m$ departing from $v_A$ still not be matched by $v_B$ in $\mathcal{I}$ since the interleaving construction of $\mathcal{I}_B$ generated from $\mathcal{I}'_B$ and $\mathcal{I}''_B$.

B. if $G'$ has a branching error state or parallel error state, it is easy to obtain there is a branching error state or parallel error state in $\mathcal{I}$.

C. Similarly for $G''$, we can obtain the same results.

Therefore, we get the proof.

4. If $G = G' + G''$, 

(a) \((\Rightarrow)\) We assume that \(G\) is well-formed, then \(G'\) and \(G''\) are well-formed as well. Let \(T = \bigotimes_{A \in G} (\overline{G_{\downarrow A}})\), \(T' = \bigotimes_{A \in G'} (\overline{G'_{\downarrow A}})\) and \(T'' = \bigotimes_{A \in G''} (\overline{G''_{\downarrow A}})\) be the product of the projections of \(G\), \(G'\) and \(G''\) respectively. By the induction hypothesis, we obtain

\[
T' = \bigotimes_{A \in G'} (\overline{G'_{\downarrow A}}) = \bigotimes_{A \in G'} (\overline{G'_{\downarrow A}}),
\]

\[
T'' = \bigotimes_{A \in G''} (\overline{G''_{\downarrow A}}) = \bigotimes_{A \in G''} (\overline{G''_{\downarrow A}}).
\]

In other words, there is no error state, no branching error states and no parallel state in \(T'\) and \(T''\).

According to Lemma 5.3.8, we have

\[
\bigotimes_{A \in G} (\overline{G_{\downarrow A}}) = \bigotimes_{A \in G'} (\overline{G'_{\downarrow A}}) + \bigotimes_{A \in G''} (\overline{G''_{\downarrow A}})
\]

\[
= \bigotimes_{A \in G'} (\overline{G'_{\downarrow A}}) + \bigotimes_{A \in G''} (\overline{G''_{\downarrow A}}).
\]

Therefore, there are no error states in \(\bigotimes_{A \in G} (\overline{G_{\downarrow A}})\) since the branching composition of two GIA will not generate error states. Therefore, we have \(\bigotimes_{A \in G} (\overline{G_{\downarrow A}}) = \bigotimes_{A \in G'} (\overline{G'_{\downarrow A}})\). And due to \(G\) being well-formed, there are at most a active participant and the rest all being passive participants. According to Definition 5.2.9, it is impossible that \(\bigotimes_{A \in G} (\overline{G_{\downarrow A}})\) have any branching error states. As well as there is no parallel error state. Then, we get the proof.

(b) \((\Leftarrow)\) We use contraposition. So the new statement is that if \(G = G' + G''\) is a not well-formed branching composition, then, we obtain \(\bigotimes_{A \in G} (\overline{G_{\downarrow A}}) \neq \bigotimes_{A \in G} (\overline{G'_{\downarrow A}})\) or there exist branching error states or parallel error state. In other words, if \(G\) is a not well-formed branching, then, there must be error states or branching error states or parallel error states in \(\bigotimes_{A \in G} (\overline{G_{\downarrow A}})\).

We distinguish the following cases (and for simplicity we consider just binary choices):

i. Suppose there is more than one active participant; say \(A\) and \(B\)
The above two structures describes the initial transitions of $\overline{\mathcal{G}_{\downarrow A}}$ and $\overline{\mathcal{G}_{\downarrow B}}$ respectively, so that, if there is at least a send action from described above will not be consumed in $\otimes_{A \in \mathcal{G}} \overline{\mathcal{G}_{\downarrow A}}$. According to Theorem 4.4.2, there must be an error state in $\otimes_{A \in \mathcal{G}} \overline{\mathcal{G}_{\downarrow A}}$. If these four send actions will be consumed in $\otimes_{A \in \mathcal{G}} \overline{\mathcal{G}_{\downarrow A}}$. There is a branching error state which has outgoing transition action with two different sender in $\otimes_{A \in \mathcal{G}} \overline{\mathcal{G}_{\downarrow A}}$.

ii. If there is a participant $A$ which is neither active nor passive then, if in the initial state $A$

A. has two send actions, say

then we proceed as in 4(b)i; if one of the send actions is not consumed in $\otimes_{A \in \mathcal{G}} \overline{\mathcal{G}_{\downarrow A}}$ then (Theorem 4.4.2) there must be an error state in $\otimes_{A \in \mathcal{G}} \overline{\mathcal{G}_{\downarrow A}}$. If both send actions are consumed in $\otimes_{A \in \mathcal{G}} \overline{\mathcal{G}_{\downarrow A}}$ then there is a branching error state which has two same outgoing transition in $\otimes_{A \in \mathcal{G}} \overline{\mathcal{G}_{\downarrow A}}$, or two branching error state as shown by case 3 in Definition 5.2.9.

B. has two receive actions, say

then by symmetry or the projection relation, there must exist a participant $B$ with two transitions $BA!m$, yielding the previous case.
C. has a \( \tau \) transition from the initial state to a state which does not have any outgoing transitions, the projection of \( A \) has one of the following shapes.

\[
\begin{align*}
&\quad \tau \\
&\quad \quad \downarrow \\
&v_0 \quad AB!m \\
&\quad \quad \downarrow \\
&v_1 \\
&\quad \quad \downarrow \\
&v_2 \\
&\quad \uparrow \\
&\quad \quad \downarrow \\
&v_3 \\
&\quad \quad \downarrow \\
&v_4 \\
&\quad \uparrow \\
&\quad \quad \downarrow \\
&v_5 \\
&\quad \quad \downarrow \\
&v_6
\end{align*}
\]

This is possible only if a participant is not involved in either branch. Therefore, this must a send actions will not be consumed in \( \otimes_{A \in G} \overline{G_{\downarrow A}} \) since \( A \) executes the transition in the branch, the other participants execute the transitions in the other branch. Therefore, there is an error state.

D. has a send and a receive action; the projection of \( A \) has one of the following shapes.

\[
\begin{align*}
&\quad \downarrow \\
&\quad \quad \downarrow \\
&v_0 \quad AB!m \\
&\quad \quad \downarrow \\
&v_1 \\
&\quad \quad \downarrow \\
&v_2 \\
&\quad \quad \Downarrow \\
&v_3 \\
&\quad \quad \Updownarrow \\
&v_4 \\
&\quad \quad \Downarrow \\
&v_5 \\
&\quad \quad \Downarrow \\
&v_6
\end{align*}
\]

As in 4(b)i, if there is a send action not consumed in \( \otimes_{A \in G} \overline{G_{\downarrow A}} \) then there is an error state \( \otimes_{A \in G} \overline{G_{\downarrow A}} \) (by Theorem 4.4.2). Then by symmetry or the projection relation, there must exist a participant \( C \) with transition \( CA!n \), yielding the previous case. If these send actions \( AB!m, CA!n \) will be consumed in \( \otimes_{A \in G} \overline{G_{\downarrow A}} \), There is a branching error state which has two outgoing transition action with different sender in \( \otimes_{A \in G} \overline{G_{\downarrow A}} \).

5. If \( G = \ast \{G'\} @ A \),

(a) \( \Rightarrow \) We assume that \( G \) is well-formed, then, \( G' \) is well-formed as well. Let \( \mathcal{I} = \otimes_{A \in G} (\overline{G_{\downarrow A}}) \) and \( \mathcal{I}' = \otimes_{A \in G'} (\overline{G_{\downarrow A}}) \) be the \( \otimes \)-product of projections of \( G \), \( G' \) respectively. In fact, we know \( G = G' \). Then by the induction hypothesis,
we obtain

\[ I' = \otimes_{A \in G'}(G_{L+A}) = \|_{A \in G'}(G_{\downarrow A}). \]

That means there are no error states, no branching error states and parallel error states in \( I' \). Then, based on Lemma 5.3.10, we are able to get

\[ I = *(I', \dot{I}_l, \dot{I}_e), \]

By Definition 5.3.9, \( I \) has no error states by construction, we know that if \( I' \) does not have error states or branching error states, \( *(I', \dot{I}_l, \dot{I}_e) \) must not have any error states, branching error states or parallel error states either.

(b) \( \leftarrow \) We use contra position to help with the proof. So the new statement is if the iteration \( G = *\{G'\}@A \) is not well-formed, we obtain \( \otimes_{A \in G'}(G_{\downarrow A}) \neq \|_{A \in G'}(G_{\downarrow A}) \). In other words, if \( G \) is not well-formed, then, there must be error states, branching error states or parallel error states in \( \otimes_{A \in G'}(G_{\downarrow A}) \).

Due to \( G = *\{G'\}@A \) being not well-formed, then we know \( G' \) must be not well-formed. In order to get the proof, we distinguish the following cases.

i. Suppose \( G' \) is not a well-formed interaction, sequential, branching or parallel, we know there must be error states, branching error states or parallel error states in \( \otimes_{A \in G'}(G_{\downarrow A}) \) from the above proofs. According Lemma 5.3.10 and Definition 5.3.9, we obtain that there must be error states, branching error states or parallel error states in \( \otimes_{A \in G'}(G_{\downarrow A}) \).

ii. Suppose \( G' \) is not a well-formed iteration, that means \( G' \) is a nested iteration, therefore, we assume \( G' = *\{G''\}@B \).

A. Suppose \( G'' \) is not a well-formed interaction, sequential, branching or parallel, similar to (5(b)i), we obtain that there must be error states, branching error states or parallel error states in \( \otimes_{A \in G'}(G_{\downarrow A}) \). Therefore, we also get there must be error states or branching error states or parallel states in \( \otimes_{A \in G'}(G_{\downarrow A}) \).
5.5 A Few Examples

We show some examples to highlights how well-formedness is checked.

5.5.1 Examples of sequential composition

Then, we are going to give two typical instances of sequential compositions of two g-choreographies. As said, we do constraint sequential composition of g-choreographies. Therefore, sometimes, some sequential compositions are similar to parallel compositions.

A well-formed sequential composition

The sequential composition $G_1 = A \xrightarrow{m} B; B \xrightarrow{n} C$ is a well-formed g-choreography constructed by connecting two interactions $A \xrightarrow{m} B$ and $B \xrightarrow{n} C$ shown in Fig. 5.16, there are causal dependencies between sequential components. We are able to see there are three participants $A$, $B$, $C$ and two messages $m$, $n$ involved in $G_1$. Then, we see there exists causal dependencies between the semantics of these two interactions, where the receive action $AB?m$ must happen before the send action $BC!n$. The sequential composition $G_1$ specifies the interaction $B \xrightarrow{n} C$ must happen after the interaction $A \xrightarrow{m} B$. Intuitively, this specification of $G_1$ is reasonable and implementable since all three participants are able to coordinate well in $G_1$.

Then, let us see the formal verification process shown in Fig. 5.17. The GIA $I_A$, $I_B$ and $I_C$ represent the projection of the participants $A$, $B$ and $C$ respectively.
5.5 A Few Examples

A few examples of sequential compositions are shown in Figs. 5.16 and 5.17. The sequential composition $G_1 = \langle A \stackrel{m}{\rightarrow} B; B \stackrel{n}{\rightarrow} C \rangle$ is well-formed.

While $I_A, I_B, I_C$ are the GIA without removable $\tau$-transitions refined from $I_A, I_B$ and $I_C$ respectively. We are able to see all the $\tau$-transitions in the projections are removable. And there is no error states existing in $I_A \otimes I_B \otimes I_C$, therefore, the sequential composition $G_1$ is well-formed.

**Another well-formed sequential composition**

The sequential composition $G_2 = \langle A \stackrel{m}{\rightarrow} B; C \stackrel{n}{\rightarrow} D \rangle$ is a well-formed $g$-choreography constructed by connecting two interactions $A \stackrel{m}{\rightarrow} B$ and $C \stackrel{n}{\rightarrow} D$ shown in Fig. 5.18, there is no causal dependencies between sequential components. Three participants $A, B, C, D$ and two messages $m, n$ are involved in $G_2$. Then, we are aware that there is no causal dependencies between the semantics of these two interactions. The sequential composition $G_2$ specifies the interaction $C \stackrel{n}{\rightarrow} D$ must happen after the interaction $A \stackrel{m}{\rightarrow} B$. Intuitively, this specification of $G_2$ seems not reason since all the participants are independent and there is no any relationships between these two interactions. The execution order can not be guaranteed. However, as we mentioned in the previous section, this kind of sequential are able to be considered as a parallel, they are equivalent semantically in fact. Therefore, we consider that $G_2$ is also well-formed.

Then, let us see the formal verification process shown in Fig. 5.19. The GIA $I_A, I_B, I_C$ and $I_D$ represent the projection of the participants $A, B, C$ and $D$ respectively. While $I_A, I_B, I_C$ and $I_D$ are the GIA without removable $\tau$-transitions refined from $I_A, I_B, I_C$ and $I_D$ respectively. In addition, we are able to see all the $\tau$-transitions in the
5.5 A Few Examples

Fig. 5.17 Verification of the well-formedness of $G_1$

$p_0 \xrightarrow{AB!m} p_1 \xrightarrow{\tau} p_2$

$\mathcal{I}_A$

$p_0 \xrightarrow{AB!m} p_1 \xrightarrow{BC!n} p_2$

$\mathcal{I}_B$

$p_0 \xrightarrow{BC!n} p_2$

$\mathcal{I}_C$

$p_0 \xrightarrow{BC!n} p_1 \xrightarrow{\tau} p_2$

$\mathcal{I}_A \otimes \mathcal{I}_B \otimes \mathcal{I}_C = \mathcal{I}_A \parallel \mathcal{I}_B \parallel \mathcal{I}_C$

$p_0 \xrightarrow{AB!m} p_1 \xrightarrow{BC!n} p_2$

$p_0 \xrightarrow{BC!n} p_1 \xrightarrow{\tau} p_2$

$\mathcal{I}_A \otimes \mathcal{I}_B \otimes \mathcal{I}_C = \mathcal{I}_A \parallel \mathcal{I}_B \parallel \mathcal{I}_C$

$p_0 \xrightarrow{AB!m} p_1 \xrightarrow{BC!n} p_2$

$p_0 \xrightarrow{BC!n} p_1 \xrightarrow{\tau} p_2$

$\mathcal{I}_A \otimes \mathcal{I}_B \otimes \mathcal{I}_C = \mathcal{I}_A \parallel \mathcal{I}_B \parallel \mathcal{I}_C$

Fig. 5.18 A well-formed sequential composition $G_2$

$G_2 = A \xrightarrow{\pi} B; C \xrightarrow{\pi} D$

projections are removable. And there is no error states existing in $\mathcal{I}_A \otimes \mathcal{I}_B \otimes \mathcal{I}_C \otimes \mathcal{I}_D$, therefore, the sequential composition $G_2$ is well-formed.
5.5 A Few Examples

Moreover, when we review the outcome of the $\otimes$-product $\mathcal{I}_A \otimes \mathcal{I}_B \otimes \mathcal{I}_C \otimes \mathcal{I}_D$, we will find the product representing the real executions of the sequential composition $G_2$ which is a interleaving of executing the interactions $A \xrightarrow{m} B$ and $B \xrightarrow{n} C$.

5.5.2 Examples of parallel composition

A well-formed parallel composition

The parallel composition $G_3 = A \xrightarrow{m} C \parallel B \xrightarrow{n} C$ shown in Fig. 5.20 is a well-formed g-choreography constructed by composing two interactions $A \xrightarrow{m} C$ and $B \xrightarrow{n} C$ as two threads. We can see there are three participants $A$, $B$, $C$ and two messages $m$, $n$.
involved in \( G_3 \). The parallel composition \( G_3 \) specifies that the two threads \( A \xrightarrow{m} B \) and \( B \xrightarrow{n} C \) are able to execute in parallel.

\[
G_3 = A \xrightarrow{m} C \parallel B \xrightarrow{n} C
\]

Fig. 5.20 A well-formed parallel composition \( G_3 \)

Intuitively, the specification of \( G_3 \) is reasonable and implementable since all three participants are able to coordinate well during the executions of the parallel. Let us see the formal verification shown in Fig. 5.21. The GIA \( I_A, I_B \) and \( I_C \) describe the projection of the participants \( A, B \) and \( C \) respectively. While \( \overline{I}_A, \overline{I}_B, \overline{I}_C \) are the GIA without removable \( \tau \)-transitions refined from \( I_A, I_B, I_C \) respectively. The parallel composition \( G_3 \) is well-formed since there is no (parallel)error states in \( \overline{I}_A \otimes \overline{I}_B \otimes \overline{I}_C \). In addition, the execution of threads in \( G_3 \) represents by the interleaving structure in \( \overline{I}_A \otimes \overline{I}_B \otimes \overline{I}_C \).

A not well-formed parallel composition

The parallel composition \( G_4 = A \xrightarrow{m} B \parallel (A \xrightarrow{m} B + A \xrightarrow{n} B) \) is not a well-formed g-choreography which is constructed by composing an interactions \( A \xrightarrow{m} B \) and a branching composition \( A \xrightarrow{m} B + A \xrightarrow{n} B \) shown in Fig. 5.22. We can see there are three participants \( A, B, C \) and two messages \( m, n \) involved in \( G_3 \). The parallel composition \( G_4 \) specifies that the thread \( A \xrightarrow{m} B \) is able to execute concurrently with the left branch \( A \xrightarrow{m} B \) or with the right branch \( A \xrightarrow{n} B \). The parallel composition \( G_4 \) is not well-formed since there is an interaction \( A \xrightarrow{m} B \) existing in both threads. So that, the right thread may steal messages sent from left thread, then may result potential deadlocks.
5.5 A Few Examples

Fig. 5.21 Verification of the well-formedness of $G_3$

Let us see the formal verification shown in Fig. 5.23. The GIA $I_A$ and $I_B$ represent the projection of the participants $A$ and $B$ respectively. While $I_A$, $I_B$ are the GIA without removable $\tau$-transitions refined from $I_A$, $I_B$ respectively. We are able to see state $v_0 u_0$ in $I_A \otimes I_B$ denoted by a blue oval is a parallel error state since $v_0 u_0$ is the vertex of six diamonds which have $AB!m$ as label of edges. As well as we are able to state $v_1 u_2$ in $I_A \otimes I_B$ denoted by a red box is an error state since $AB!n$ at $v_1$ is unmatched by $u_2$. Basically, in $G_4$, the message $m$ sent by left thread is stolen by the right thread, then when the right thread intends to send message $n$, it causes a
situation that $n$ never be consumed. Therefore, the parallel composition $G_4$ is not well-formed since there are both error states and parallel error states in $\mathcal{I}_A \otimes \mathcal{I}_B$.

5.5.3 Examples of branching composition

A well-formed branching composition

The branching composition $G_5 = A \xrightarrow{m} B; B \xrightarrow{x} C + A \xrightarrow{n} C; C \xrightarrow{y} B$ is a well-formed $g$-choreography with an active participant $A$ and two passive participants $B, C$. The branching composition $G_5$ is constructed by composing two sequential compositions, where the left branch is $A \xrightarrow{m} B; B \xrightarrow{x} C$ and the right branch is $A \xrightarrow{n} C; C \xrightarrow{y} B$ shown in Fig. 5.24. There are three participants $A, B, C$ and four messages $m, n, x, y$ involved in $G_5$. The branching composition $G_5$ specifies that participant $A$ decide to execute one of the branches (the left one or the right one). In other words, participant $A$ makes a non-deterministic choice to run $A \xrightarrow{m} B; B \xrightarrow{x} C$ or $A \xrightarrow{n} C; C \xrightarrow{y} B$.

We consider the specification of $G_5$ is reasonable and implementable since just participant $A$ are able to autonomously make the non-deterministic choice from the
branching starting point. Participant A determines which messages and participants are able to be interacted. If A decides to execute \( A \xrightarrow{m} B \), afterwards, participant B have to execute \( B \xrightarrow{x} C \). Then, all the participants know \( G_5 \) choose the left branch. All the participants will be terminated successfully.

Let us check the formal verification shown in Fig. 5.25. The GIA \( \mathcal{I}_A, \mathcal{I}_B \) and \( \mathcal{I}_C \) are the GIA representing the projection of the participants A, B and C respectively. While \( \overline{\mathcal{I}}_A, \overline{\mathcal{I}}_B, \overline{\mathcal{I}}_C \) are the GIA without removable \( \tau \)-transitions refined from \( \mathcal{I}_A, \mathcal{I}_B \) and \( \mathcal{I}_C \) respectively. Then, we are able to see neither error states nor branching, parallel error states exists in \( \overline{\mathcal{I}}_A \otimes \overline{\mathcal{I}}_B \otimes \overline{\mathcal{I}}_C \). Therefore, this branching composition is well-formed. Additionally, there is an error state appearing in the intermediate step \( \overline{\mathcal{I}}_A \otimes \overline{\mathcal{I}}_B \), but it does not effect the final result. As a matter of fact, this situation which error states appear in intermediate steps happens very often. The final \( \otimes \)-product are able to avoid the execution paths which has error states.

**A not well-formed branching composition**

The branching composition \( G_6 = A \xrightarrow{m} B + A \xrightarrow{n} C \) is not a well-formed g-choreography which has an active participant A and two undefined participant B, C. The branching composition \( G_6 \) is constructed by composing two interactions \( A \xrightarrow{m} B \) and \( A \xrightarrow{n} C \) shown in Fig. 5.26. There are three participants A, B, C and two messages \( m, n \) involved in
5.5 A Few Examples

Fig. 5.25 Verification of the well-formedness of $G_5$

$G_6$. The branching composition $G_6$ specifies that from the branching starting point, participant $A$ decide to execute one of the branches (the left one or the right one).

$$G_6 = A \xrightarrow{m} B + A \xrightarrow{m} C$$

Fig. 5.26 A not well-formed branching composition $G_6$
Intuitively, the specification of $G_6$ is not reasonable and not implementable since participant $B$ does not exist in the right branch and participant $C$ does not exist in the left branch. If $A$ decides to execute $A \xrightarrow{m} B$, participant $C$ has no idea about what happens. Participant $C$ waits for the interaction from $A$ forever and does not terminate. Or, participant $B$ consider that $A$ will send messages to $C$, so $B$ terminates first, then, when participant $A$ send messages to $B$, the messages will never be consumed. In other words, participant $A$ knows which branch the $G_6$ will execute, whereas participant $B$ and $C$ have no idea about the decisions.

Let us see the formal verification shown in Fig. 5.27. The GIA $\mathcal{I}_A$, $\mathcal{I}_B$ and $\mathcal{I}_C$ describe the projection of the participants $A$, $B$ and $C$ respectively. While $\overline{\mathcal{I}}_A$, $\overline{\mathcal{I}}_B$, $\overline{\mathcal{I}}_C$ are the GIA without removable $\tau$-transitions refined from $\mathcal{I}_A$, $\mathcal{I}_B$ and $\mathcal{I}_C$ respectively. We are able to see the $\tau$-transitions in projections $\mathcal{I}_B$, $\mathcal{I}_C$ are not removable since the $\tau$-transitions will change the non-deterministic choices from $u_0$ and $w_0$. The unremovable $\tau$-transitions are able to execute autonomously without any interferences. Therefore, there is an error state (shown by red box) in $\mathcal{I}_A \otimes \mathcal{I}_B$ since $\mathcal{I}_B$ can terminate via executing the $\tau$, whereas $\overline{\mathcal{I}}_A$ has no idea about that, still try to send message $m$ to
I\_B$, it causes a situation which the message $m$ will stay in the buffer forever and not be consumed eventually.

**Another not well-formed branching composition**

The branching composition $G_7 = A \xrightarrow{m} C; B \xrightarrow{x} C + B \xrightarrow{n} C; A \xrightarrow{y} C$ is not a well-formed branching composition which has two active participants $A, C$ and a passive participant $C$. The branching composition $G_7$ is constructed by two sequential compositions as branches, where the left branch is $A \xrightarrow{m} C; B \xrightarrow{x} C$ and the right branch is $B \xrightarrow{n} C; A \xrightarrow{y} C$ shown in Fig. 5.28. There are three participants $A, B, C$ and four messages $m, n, x, y$ involved in $G_7$.

![Diagram of G7]

$G_7 = A \xrightarrow{m} C; B \xrightarrow{x} C + B \xrightarrow{n} C; A \xrightarrow{y} C$

Fig. 5.28 A not well-formed branching composition $G_7$

Intuitively, the specification of $G_7$ is not reasonable and not implementable since participant both $A$ and $B$ are able to autonomously send messages to $C$ at the beginning. Let us consider this circumstance, if $C$ has received the message $m$ from $A$, then $C$ will wait for the message $x$ sent from $B$, however, participant $B$ have no idea about that. If $B$ decide to send $x$ to $C$, this is fine. But if $B$ decide to send $n$ to $C$, it causes a situation where the message $n$ will stay in the buffer forever and not be consumed eventually.

Let us view the formal verification shown in Fig. 5.29. The GIA $I_A, I_B$ and $I_C$ represent the projection of the participants $A, B$ and $C$ respectively. While $I_A, I_B, I_C$ are the GIA without removable $\tau$-transitions refined from $I_A, I_B$ and $I_C$ respectively.
We are able to see all the \( \tau \)-transitions in the projections are removable. Additionally, there are a green diamond node and two red boxes existing in \( \mathcal{I}_A \otimes \mathcal{I}_B \otimes \mathcal{I}_C \). The state \( v_0u_0w_0 \) is a branching error state since there exist two leaving transitions which have different subjects \( A \) and \( B \). It reflects that there are two participants trying to autonomously send messages at the branching starting point of \( G_7 \). The states \( v_1u_0w_1 \) and \( v_0u_1w_2 \) are error states, they well reflect the circumstances mentioned in last paragraph.
Summary of branching compositions instances

At last, let us summarise the previous three branching composition. Firstly, for \( G_5 = A \xrightarrow{m} B; B \xrightarrow{\tau} C + A \xrightarrow{n} C; C \xrightarrow{\eta} B \) shown in Fig. 5.24, it is a perfect branching composition which has just one active participant \( A \), two passive participants \( B \) and \( C \). All the participants are able to coordinate well in communications without deadlock.

Secondly, for \( G_6 = A \xrightarrow{m} B + A \xrightarrow{n} C \) shown in Fig. 5.26, it seems that the unwell-formedness is caused by unremovable \( \tau \), while in essence, the undefined participant \( B, C \) result in the unwell-formedness. Participant \( B \) does play any roles in the right branch, whilst \( C \) does not play any roles in the left branch either. So that participants \( B \) and \( C \) do have enough information in both branches. In other words, information asymmetry leads to deadlock in communications.

Finally, for \( G_7 = A \xrightarrow{m} C; B \xrightarrow{\tau} C + B \xrightarrow{n} C; A \xrightarrow{\eta} C \) shown in Fig. 5.28, the unwell-formedness is caused by two active participants. More than one active participant means that two or many can send messages autonomously and independently at the beginning. More than one active participant may get trouble and make a mess in communications. The error states and a branching error state in \( \overline{I_A} \otimes \overline{I_B} \otimes \overline{I_C} \) reflect the incorrect non-deterministic choices.

5.5.4 Examples of iteration

The iteration \( G_8 = * \{ A \xrightarrow{m} B; A \xrightarrow{n} C \} @ A \) is a well-formed g-choreography constructed by repeating a sequential composition \( A \xrightarrow{m} B; A \xrightarrow{n} C \) shown in Fig. 5.30. There are three participants \( A, B, C \) and two messages \( m, n \) involved in \( G_8 \). The iteration \( G_8 \) specifies that repeating execution of the sequential composition \( A \xrightarrow{m} B; A \xrightarrow{n} C \) till the participant \( A \) decides to break the iteration to terminate.

Intuitively, the specification of \( G_8 \) is reasonable and implementable since the enclosed sequential composition is well-formed. Then, let us see the formal verification shown in Fig. 5.31. The GIA \( \overline{I_A}, \overline{I_B} \) and \( \overline{I_C} \) represent the projection of the participants \( A, B \) and \( C \) respectively. While \( \overline{I_A}, \overline{I_B}, \overline{I_C} \) are the GIA without removable \( \tau \)-transitions
refined from $\mathcal{I}_A, \mathcal{I}_B$ and $\mathcal{I}_C$ respectively. We consider that $G_8$ is well-formed since there is no error states in $\mathcal{I}_A \otimes \mathcal{I}_B \otimes \mathcal{I}_C$. The structure of $\mathcal{I}_A \otimes \mathcal{I}_B \otimes \mathcal{I}_C$ represents the iteration in $G_8$. 

Fig. 5.30 A well-formed iteration $G_8$
Fig. 5.31 Verification of the well-formedness of $G_8$
Chapter 6

A Prototype for GIA

The theory presented in the previous chapters is amenable of implementation. This chapter discusses a prototype that realises the basic elements of our constructions. We then discuss two simple examples.

The prototype implementation of GIA is available at https://github.com/haomoons/GIA.

6.1 An Implementation of GIA

We are developing GIA (after GIA for global graphs); the implementation is in a preliminary state; our goal is to eventually integrate GIA with ChorGram [35, 37], a tool suite for choreographic development from which GIA is written in Haskell, hence GIA is written in Haskell to facilitate its integration with ChorGram from which GIA imports the main data types. In particular, the data type to represent g-choreographies is as the one in ChorGram:

\[
\text{data } G = \text{Emp} \mid \text{Act}(\text{Ptp}, \text{Ptp}) M \mid \text{Seq}[G] \mid \text{Par}[G] \mid \text{Bra}(\text{Set } G)
\]

where \text{Ptp} and \text{M} are the types for participants and messages respectively and they are both aliases of the string type of Haskell. The empty g-choreography 0 is represented
by the constructor Emp, for interactions we use the constructor Act that takes as parameters the pair of participants (sender and receiver) and the message (so that interaction \( A \xrightarrow{m} B \) is represented by the Haskell term \( \text{Act} \("A" \ "B"" \ "m"\)\), and the remaining constructors generalise the binary sequential, parallel, and branching composition respectively with constructors Seq, Par, and Bra on lists of g-choreographies. For instance, \( G_1; G_2; G_3 \) is represented as \( \text{Seq} \ [g_1, g_2, g_3] \) if \( g_1, g_2, \) and \( g_3 \) are the representations of \( G_1, G_2, \) and \( G_3 \) respectively. Finally, GIAGG also imports the representation of states of ChorGram, namely the type State of states which, for simplicity, is another alias of Haskell’s string type.

The main files of the prototype are

**The file GG2GIA.hs** contains the projection operation implemented by the Haskell function proj2Gia. This is a recursive function taking five parameters: a g-choreography, a participant to be projected onto, an initial state, a state for composing the result with other GIA, and an integer counter used to generate fresh states.

**The file GIA.hs** contains the definition of GIA and the implementation of all the operations on GIA. A GIA is a 5-tuple defined as follows

\[
\text{type Gia} = (\text{Set State}, \text{State}, \text{Set Ptp}, \text{Set Interface}, \text{Set Tran})
\]

where Interface and Tran are the types for interfaces and transitions of GIA respectively. More precisely, Interface is a 3-tuple defined as

\[
\text{type Interface} = (\text{Channel}, \text{Dir}, \text{Message})
\]

where Channel is a pair of participants (Ptp), Dir is an enumerated type for the type of action (for our purposes we consider Send, Receive, and Sync only) type.
6.2 Applying GIAGG

Channel and Dir are defined as follows

\[
\text{data Dir} = \text{Send} \mid \text{Receive} \mid \text{Synch},
\]

where Message is (an alias for) string type of Haskell Send, Receive and Synch represent in Haskell the symbols “!”,”?” and “!?” used in GIA transitions.

The type Tran is defined as

\[
\text{type Tran} = (\text{State}, \text{Event}, \text{State})
\]

with

\[
\text{data Event} = \text{Interf Interface} \mid \text{Tau}.
\]

The three main functions are refineGia, otimesProd and findErrState.

- **refineGia** takes a GIA in input and returns the GIA without removable $\tau$-transitions
- **otimesProd** computes the $\otimes$-product between two GIA
- **findErrState** accepts two GIA as inputs and returns a list of error states.

The current state of GIAGG has a command-line interface and does not feature a GUI.

6.2 Applying GIAGG

We give an example of how the functionalities provided by GIAGG can be used. Let us consider the g-choreographies G and G’ in Fig. 6.1. Their GIAGG representation is respectively given by the Haskell terms $g$ and $g’$ defined as
6.2 Applying GIAGG

![Diagram of two g-choreographies G and G']

$G = A \xrightarrow{m} B; B \xrightarrow{n} C + A \xrightarrow{a} C; C \xrightarrow{y} B$

$G' = A \xrightarrow{m} B + A \xrightarrow{a} C$

Fig. 6.1 Two g-choreographies G and G'

$g = \text{Bra} \left( \text{Seq} \left[ \text{Act}("A", "B") "m", \text{Act}("B", "C") "x" \right], \text{Seq} \left[ \text{Act}("A", "C") "n", \text{Act}("C", "B") "y" \right] \right)$

$g' = \text{Bra} \left( \text{S.fromList} \left[ \text{Act}("A", "B") "m", \text{Act}("A", "C") "n" \right] \right)$

Accordingly, the projection $G_A$, for instance, can be computed by invoking the function $\text{proj2Gia} \ g "A" "0" "e" 5$. This invocation yields the GIA

$a = \text{S.fromList}["0", "1", "2", "3"], "0", \text{S.fromList}["A"], \text{S.fromList}[("A", "B"), \text{Send}, "m"), ("A", "C"), \text{Send}, "n"], \text{S.fromList}["0", \text{Interf} ("A", "B"), \text{Send}, "m"), "1"), ("1", \text{Tau}, "3"), ("0", \text{Interf} ("A", "B"), \text{Send}, "n"), "2"), ("2", \text{Tau}, "3")]

which represents the GIA $I_A$ in Fig. 6.2 obtained projecting $G$ on $A$ from Fig. 6.1.

Likewise, we are able to get $b, c$ corresponding to $I_B, I_C$ in Fig. 6.2 respectively. Then, we can obtain the refined GIA $ra$ corresponding to $I_A$ in Fig. 6.3 by

$ra = \text{refineGia} \ a = (\text{S.fromList}["0", "1"], "0", \text{S.fromList}["A"], \text{S.fromList}[("A", "B"), \text{Send}, "m"), ("A", "C"), \text{Send}, "n"], \text{S.fromList}["0", \text{Interf} ("A", "B"), \text{Send}, "m"), "1"), ("0", \text{Interf} ("A", "B"), \text{Send}, "n"), "1")$
Similarly, we obtain refined GIA $rb$, $rc$ from $b$, $c$ respectively. After that, we are able to obtain the $\otimes$-product between $ra$ and $rb$ by $\otimesProd ra rb$, the $\otimes$-product between $\otimesProd ra rb$ and $rc$ by $\otimesProd (\otimesProd ra rb) rc$ as well as obtain the list to verify each state in the $\otimes$-product

\[
\text{findErrState } ra rb = [\text{("0",0", False), ("1",1", False), ("1",0", False), ("1",2", False),}
\]
\[
  ("0",2", True))]
\]
\[
\text{findErrState } (\otimesProd ra rb) rc = [\text{("0",0", False), ("1",1", False), ("1",0", False), ("1",2", False),}
\]
\[
  ("1",2",2", False)].
\]
There is no error state shown in `findErrState (otimesProd ra rb) rc`. On the other hand, when we do the same computation on $G'$ in Fig. 6.1 with the invocation `findErrState (otimesProd ra' rb') rc'` we obtain an error state as expected since $G'$ is not well-formed.
Summary  We established a new way to check well-formedness of global graphs introducing the novel class of group interface automata. More precisely, we adopted a variants of the semantics of g-choreographies presented in [24, 51]. We decided to relax the notion of well-formedness by dropping the condition of well-sequencedness (so that in our theory the sequential composition is defined provided that its components have semantics). As explained in Chapter 5 (page 74) this is a design decision that we took for simplicity. We borrowed the other conditions of well-branchedness, namely well-forkedness and well-branchedness, from [24, 51]. Notice that we extended well-formedness to iterative g-choreographies in a straightforward way similar to what done in [21].

Finally, we developed the prototype implementation GIAGG to support the analysis of well-formedness of g-choreographies based on GIA. We only introduced part of the functionalities of the tool. Currently, we are implementing the functions to detect parallel and branching error states to fulfil GIAGG. Apart from empty, interactions, sequential compositions, branching compositions and parallel compositions, g-choreographies also have iterations which allow looping executions inside. Therefore, we plan to extend our tool to also facilitate iterations. In addition, we plan to extend GIAGG to the general well-formedness of g-choreographies as well. And finally, integrate all the functionalities
of GIAGG into ChorGram [37] in the future, where ChorGram is a comprehensive tool helping the design, the construction and the analysis of g-choreographies.

**Extension: general well-formedness** We remark that, besides the notion of well-branchedness adopted here, a more general one [24, 51] has been reviewed in Section 3.3.4 on page 33. The difference is that in the simpler notion the check if participants are passive or active is driven by the syntax. For instance, Definitions 3.3.7 and 3.3.8 consider the minimal actions performed by a participant $A$ at the syntactic branching point. For instance, in Fig. 7.1, the branching composition $G$ is not well-formed: participants $A$ and $B$ are neither active nor passive participants because $A$ and $B$ behave exactly same in both branches, so their minimal sets of actions are not disjoint. The general notion of well-branchedness instead consider “semantic” branching point by factoring out the “prefixes” of a participant that are common to all branches. This definition makes g-choreographies like the one in Fig. 7.1 well-branched, however it is quite involved since the notion of “prefixes” is defined on pomsets. In fact, the identification of active and passive participants relies on divergence points imposed by prefix-maps of g-choreographies instead of using the syntax. In other words, the well-branchedness allows one to identify, for each participant, where (if at all) the participant becomes aware of the choice even if this happens after some actions common to all branches. Hence, using prefix maps the choice $G$ of Fig. 7.1 is general well-branched since both $A$ and $B$ behaves differently after real divergence points in the branches, where $A$ is passive, $B$ are active and therefore aware of the choice which is indeed a “semantic” choice.

We have considered both notions and generalised our results also to the more general well-branchedness. However, we decided not to include this result in the thesis for two reasons. The first is that opted for simplicity; the second reason is that the treatment of the general notion of well-branchedness with GIA requires a substantial amount of machinery that would have made the size of the thesis rather large.
We give here an intuition of how we deal with the general notion of well-branchedness using some examples. Let us consider the g-choreography $G$ in Fig. 7.1 with the GIA $\Delta I_A$ and $\Delta I_B$ in Fig. 7.2. We remark that $G$ is general well-formed (since, as we explained above, $B$ is active and $A$ is passive) but not well-formed (since the initial actions of $A$ and $B$ on the two branches are not disjoint). The main difference between $I_A$ and $I_B$ in Fig. 7.1 and $\Delta I_A$ and $\Delta I_B$ in Fig. 7.2 is that the latter pair of GIA is obtained by determinising the pair of projections $I_A$ and $I_B$ of $G$. Note that there is no error state in the product $\Delta I_A \otimes \Delta I_B$ (cf. the right-most GIA in Fig. 7.2). However, this is not sufficient to guarantee the general well-formedness of $G$, as shown in the next example. The g-choreography $G'$ in Fig. 7.3 is not general well-formed since participants $A$ and $C$ are both active. The product of the determinised projections of $G'$ $\Delta I_A$, $\Delta I_B$, $\Delta I_C$, and $\Delta I_D$ does not contain any error states (cf. the last GIA in Fig. 7.3).
Besides the absence of error states, the check of general well-branchedness requires the language equivalence between g-choreography and its projections corresponding GIA. Let us consider again the example in Fig. 7.1; the runs of g-choreography $G$ is the set of maximal sequences of synchronisations of $G$, namely

$$\text{run}(G) = \{AB?!m \cdot BA?!x, AB?!m \cdot BA?!y\}$$

where the symbol $\_ \cdot \_$ is concatenation operation. In other words, the set of traces emerging from each branch. This set equals the (maximal) traces $\Delta I_A \otimes \Delta I_B$ in Fig. 7.2. Hence, the distributed executions are “consistent” with the ones specified in $G$. 

Fig. 7.3 A not general well-branchedness instance
Now let us consider the runs of g-choreography $G'$ in Fig. 7.3:

$$\text{run}(G') = \{AB!!m \cdot CD?!x, CD?!x \cdot AB!!m, CD?!y \cdot AB!!n, AB!!n \cdot CD?!y\}$$

We have that $\text{run}(G')$ differs from the set of (maximal) traces of $\Delta I_A \otimes \Delta I_B \otimes \Delta I_C \otimes \Delta I_D$ in Fig. 7.3, that is the set

$$\{AB!!m \cdot CD?!x, CD?!x \cdot AB!!m, CD?!y \cdot AB!!n, AB!!n \cdot CD?!y, AB!!m \cdot CD?!y, AB!!n \cdot CD?!x, CD?!y \cdot AB!!m, CD?!x \cdot AB!!n\}.$$  

Hence, $G'$ is not well-branched in the general sense because its projections provide "more" executions when running distributively.

To sum up, the verification of the general well-formedness requires the absence of error states in the determinised projections as well as that the g-choreography has the same (maximal) runs of its projections.

**Extension: repairing non well-formedness** We now discuss how GIA may be used to address a different problem: is it possible to support the design of well-formed choreographies? This problem was first discussed in [7, 6] for a different choreographic framework featuring a design-by-contract approach [5]. We conjecture that GIA may help to address the problem of "repairing" non well-formed g-choreographies following the pattern in Fig. 7.4.

![Fig. 7.4 Research conjecture on repairing g-choreographies](image)

Roughly speaking, (1) projection yields a set of GIA corresponding to each local participant involved in the g-choreography; (2) the $\otimes$-product among GIA may detect...
the error states, then, the GIA need to be repaired/refined according to the error states; (3) we transform repaired GIA into CFSMs and (4) we apply the synthesis algorithm of [36] to obtain a well-formed g-choreography. The automatic repairing of choreographies can be achieved via the processes described in Fig. 7.4. As a matter of fact, we transfer the repairing work from focusing g-choreography to GIA. So that, how to repair GIA becomes the most challenging work. We conjecture that the notations of the refinement [13, 14] for interface automata will be adopted to repair GIA.
References


