Interface Automata for Choreographies

Hao Zeng

Department of Informatics
University of Leicester
Leicester, UK

hz110@le.ac.uk

Alexander Kurz

Chapman University
California, USA

akurz@chapman.edu

Emilio Tuosto

Gran Sasso Science Institute, IT and
Department of Informatics
University of Leicester, UK

emilio.tuosto@gssi.it

Choreographic approaches to message-passing applications can be regarded as an instance of the model-driven development principles. Choreographies specify interactions among distributed participants coordinating among themselves with message-passing at two levels of abstractions. A *global view* of the application is specified with a model that abstracts away from asynchrony while a *local view* of the application specifies the communication pattern of each participant. Noteworthy, the latter view can typically be algorithmically obtained by *projection* of the global view. A crucial element of this approach is to verify the so-called *well-formed* conditions on global views so that its projections realise a sound communication protocol. We introduce a novel local model, group interface automata, to represent the local view of choreographies and propose a new method to verify the well-formedness of global choreographies. We rely on a recently proposed semantics of global views formalised in terms of pomsets.

1 Introduction

Nowadays distributed applications are widely used in our daily life, ranging from online payments via social communications and web services to multi-core computing. The engineering of distributed systems inevitably entwines with communication among components, which is a key element for realising distributed coordination mechanism. The formalisation of coordination protocols becomes an important challenge. Indeed the use of abstract models to tame the complexity of distributed applications is becoming commonplace also in industrial context [1].

To this end, *choreographies* [10] have been proposed as a methodology to facilitate the coordination of distributed components. Roughly speaking, a choreography is made of two elements, the *global view* and the *local view* [10]. The former specifies the interactions among distributed components in terms of a so-called application level protocol, where the behaviour of the system is described in terms of the relations among the (*role* of each) component. The local view yields a more concrete specification whereby the behaviour of each component "in isolation" is derived from its role in the global view; at this level of abstraction, components can be thought of as autonomous agents enacting a role described in the global view, *regardless of* the other components.

In this paper, we are interested in message-passing systems and we adopt a variant of *g-choreographies* (after *global choreographies*) [7, 8] as a formalisation of global
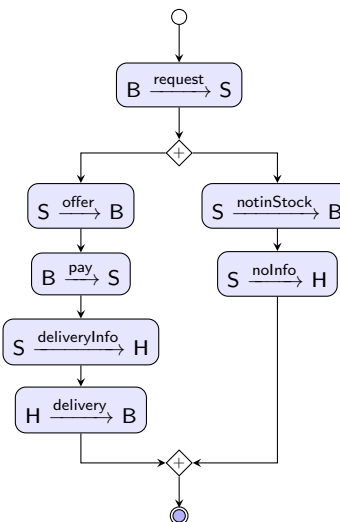

Figure 1: A simple g-choreography

views of this application domain. A main motivation to borrow g-choreographies is that they are rather expressive (see the discussion in [7, 8]) and have an abstract semantics based on pomsets. For instance, g-choreographies abstract away from asynchrony (interactions are atomic and not specified in terms of send/receive actions of participants). In this way, g-choreographies simplify the description of the system by relegating the complexity of asynchronous communication in the local views. We illustrate the model through an online shopping scenario given in Figure 1. Buyer B sends a request to Seller S, asking for some goods. After the reception of this message, S decides to send (*i*) an offer back to B or (*ii*) to a notin-Stock message to B depending on the availability of the goods. In the first case, B sends a message pay to S representing a payment. After receiving the online payment, S sends the deliveryInfo to a shipper H. Finally, H sends a delivery message to B. If the alternative (*ii*) is taken, S sends a noInfo message to H notifying that this purchase is not successful. Figure 1 gives a visual description of this g-choreography. 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.

We opt for local views of choreographies describing the behaviour of each participant in terms of send and receive actions. More precisely, we introduce a class of interface automata [3, 4] and show how they can be used to check the correctness of a model of global specifications. In our formal framework, the local views of each participant are obtained by projections from a g-choreography. In fact, the relationship between g-choreography and local views can be represented by following diagram:

$$\text{G-choreography} \xrightarrow{projection} \text{Interface automata} \xleftarrow{comply} \text{Local System}$$

Here we neglect the *comply* relation and focus only on the *projection* one. In particular, we show how our interface automata can be used to check the *well-formedness* of g-choreographies.[1] Technically, we define an internal product 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 extend interface automata [3, 4] by defining *group interface automata*. The group interface automaton $\mathscr{I}_B$ for the local view of buyer B of Figure 1 is
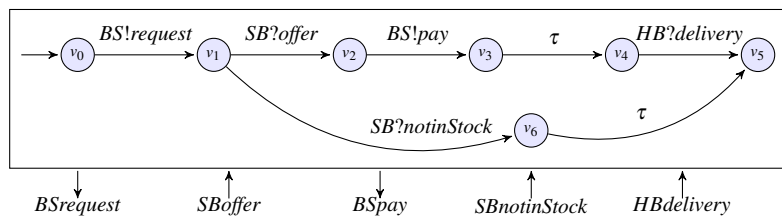
Roughly speaking, the rectangle specifies the scope of the machine, the down and up arrows on the edge of the rectangle describe the output and input interfaces respectively (for more details see section 3).

Main contributions   This paper is based on the choreographic framework presented in [7, 8]. We elaborate on the pomset semantics for global specifications (g-choreographies) given in [7, 8] and formalising 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 [3]. The notion of well-formedness identifies a sufficient condition to guarantee that

---

[1]As known from the literature, well-formedness of g-choreographies guarantees well-behaviour of local components projected from the global specification (see e.g. [9, 11, 7, 8]) .

the asynchronous execution of the projections of a g-choreographies is sound, that is the execution is deadlock-free and without orphan messages or unspecified receptions [2].

The main contribution of this paper is the reduction of well-formedness of g-choreographies to the absence of *error-states* in a variant of interface automata. More precisely we show that a g-choreography G is well-formedness 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 extend 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.

Related work   We review the choreographic models that are closest to the model presented in this paper. *Global graphs* were introduced in [5] as graphical interpretation of *global types* [9] and then refined in [7, 8] as g-choreograhies. This paper uses a variant of g-choreographies (cf. Definition 2.10) and their pomset semantics as the global view of choreographies. The variant disregards the requirements of well-sequencedness imposed in [7, 8], uses well-forkedness in [7, 8] and a relaxed notion of well-branchedness
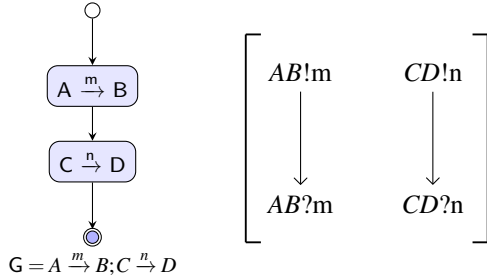
for simplicity. For instance, a sequential composition G with its pomset semantics in our variant are shown on the left figure. According to [7, 8], the semantics of G is undefined since G violates well-sequencedness due to the lack of causal dependencies among the participants of the two interactions. Instead, our variant gives G the semantics represented by the pomset shown in the right-hand-side of the figure, which simply allows $A \xrightarrow{m} B$ and $C \xrightarrow{n} D$ to run concurrently. Therefore, the semantics of G is equivalent to the semantics of the parallel composition $A \xrightarrow{m} B \mid C \xrightarrow{n} D$.

Next, we review the local models. *Interface automata* [3, 4] 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 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)* [2] are a convient 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 a notion of special internal τ-transitions, and redefine products and error states. These changes allow us to adopt interface automata to anaylse global choreographies (for details see sections 3 and 4).

2   Background

This section summarises the main concepts we use in the paper. We adapt the definitions from [7]. We write $\mathscr{P}$ for a set of participants and $\mathscr{M}$ for a set of messages.

Definition 2.1 (Global Choreography). *A global choreography (g-choreography for short) is a term G derived by the grammar*

$$G ::= 0 \mid A \xrightarrow{m} B \mid G; G' \mid G|G' \mid G + G'$$

The empty g-choreography is 0; $A \xrightarrow{m} B$ is an interaction where message $m \in \mathcal{M}$ is sent from participant $A \in \mathscr{P}$ to participant $B \in \mathscr{P}$ ($A \neq B$); the operators $\_;\_$ and $\_|\_$ and $\_+\_$ allow us to compose g-choreographies sequentially, in parallel and in non-deterministic branches. The corresponding visual notation of g-choreographies is in Figure 2. Circled and double circled nodes represent the unique initial and terminal node of each g-choreography.
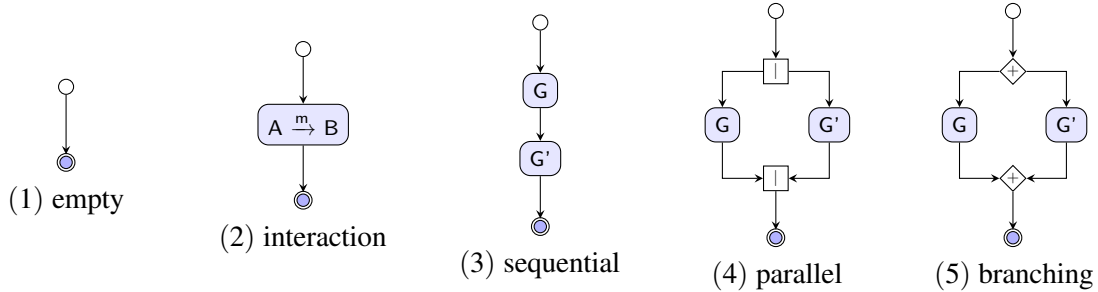

Figure 2: The visual notations of g-choreographies

Pomsets for g-choreographies   In [7] g-choreographies have been equipped with a pomset semantics establishing causal dependencies among communication actions on channels. Formally, the set of channels is $\mathscr{C} = \{(A,B) \mid A, B \in \mathscr{P}, A \neq B\}$ (and we abbreviate $(A,B) \in \mathscr{C}$ as AB). The set $\mathscr{L}$ of labels

$$\mathscr{L} = \mathscr{L}^! \cup \mathscr{L}^? \text{ where } \mathscr{L}^! = \mathscr{C} \times \{!\} \times \mathscr{M} \text{ and } \mathscr{L}^? = \mathscr{C} \times \{?\} \times \mathscr{M}.$$

consists of the elements in the set $\mathscr{L}^!$ of outputs representing send actions and $\mathscr{L}^?$ is the set of inputs representing receive actions. We abbreviate $(AB, !, m) \in \mathscr{L}^!$ and $(AB, ?, m) \in \mathscr{L}^?$ with $AB!m$ and $AB?m$ respectively.

Definition 2.2 (Lposet [7]). *The subject sbj(_) and the object obj(_)of an action are defined by*

$$sbj(AB!m) = A \quad sbj(AB?m) = B \qquad and \qquad obj(AB!m) = B \text{ and } obj(AB?m) = A$$

A labelled partially ordered set r for g-choreographies (lposet) is a triple $(\mathscr{E}, \leq, \lambda)$, with $\mathscr{E}$ a set of events, $\leq \subseteq \mathscr{E} \times \mathscr{E}$ a partial order on $\mathscr{E}$, and $\lambda : \mathscr{E} \to \mathscr{L}$ a labelling function.

Note that Definition 2.2 permits to have $e \neq e'$, $\lambda(e) = \lambda(e')$, namely two events occur in different places with the same action. The relation $\leq$ is a partial order representing the causal dependencies among events. We use $e \to e'$ to denote $e \leq e'$. Also ε denotes the empty lposet.

Definition 2.3 (Isomorphism of lposets [7]). *Two lposets $(\mathscr{E}, \leq, \lambda)$ and $(\mathscr{E}', \leq', \lambda')$ are isomorphic iff there exists a bijection $\phi : \mathscr{E} \to \mathscr{E}'$ such that $e \leq e'$ iff $\phi(e) \leq' \phi(e')$ and $\lambda = \lambda' \circ \phi$.*

Definition 2.4 (Pomset [7]). *A partially-ordered multi-set $[\mathscr{E}, \leq, \lambda]$ (of actions), pomset for short, is the isomorphism class of an lposet $(\mathscr{E}, \leq, \lambda)$.*

The advantage of using pomsets to formalise the semantics of g-choreographies is that partial orders explicitly represent the causal dependencies among communications.

Given a basic interaction $\mathsf{G} = A \xrightarrow{m} B$, the pomset $\llbracket \mathsf{G} \rrbracket$ of G is

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

Pomsets also have visual notation

$$\llbracket A \xrightarrow{m} B \rrbracket = \left[AB!m \to AB?m \right]$$

where the pair of square brackets specifies the border of the pomset, the arrow from $AB!m$ to $AB?m$ shows the happen-before relationship between events.

We now start to review the pomset semantics of sequential and parallel composition. Given a natural number n, $\mathbf{n}$ represents the singleton $\{n\}$ and $X \uplus Y$ the disjoint union of two sets X and Y. Also, given a function f on X, we let $f \otimes \mathbf{n}$ be the function extending f to $X \times \mathbf{n}$ mapping (x, n) to $f(x)$; 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 2.5 (Pomsets for sequential compositions). *Let* $r = [\mathscr{E}, \leq, \lambda]$ *and* $r' = [\mathscr{E}', \leq', \lambda']$ *be two pomsets. For a pomset r and a participant $A \in \mathscr{P}$, let $\mathscr{E}_{r,A} = \{ e \in \mathscr{E}_r \mid sbj(\lambda_r(e)) = A \}$ be the set of events of A in $\mathscr{E}_r$. The semantics of the sequential composition $seq(r, r')$ of r and r' is defined as*

$$seq(r, r') = [\mathscr{E} \uplus \mathscr{E}', \leq_{seq}, (\lambda \otimes \mathbf{1}) \cup (\lambda' \otimes \mathbf{2})],$$

where

$$\leq_{seq} = \left((\leq \otimes \mathbf{1}) \cup (\leq' \otimes \mathbf{2}) \cup \bigcup_{A \in \mathscr{P}} ((\mathscr{E}_{r,A} \times \mathbf{1}) \times (\mathscr{E}_{r',A} \times \mathbf{2})) \right)^{\star}$$

and $\star$ is the reflexive-transitive closure.

Now, we review parallel compositions of pomset.

Definition 2.6 (Pomsets for parallel compositions). *Let $r = [\mathscr{E}, \leq, \lambda]$ and $r' = [\mathscr{E}', \leq', \lambda']$ be two pomsets. The semantics of the parallel composition $par(r, r')$ of r and r' is defined as*

$$par(r, r') = [\mathscr{E} \uplus \mathscr{E}', (\leq \otimes \mathbf{1}) \cup (\leq' \otimes \mathbf{2}), (\lambda \otimes \mathbf{1}) \cup (\lambda' \otimes \mathbf{2})].$$

Roughly speaking, the sequential composition of pomsets r and r' adds causal dependencies between the communication events of r and r' done by the same participant while parallel composition does not.

Well-formedness We relax the notion of well-formedness of g-choreographies given in [7, 8] by considering only *well-forkedness* and *well-branchedness* [7, 8]:

Definition 2.7 (well-forkedness). *Pomsets $r = [\mathscr{E}, \leq, \lambda]$ and $r' = [\mathscr{E}', \leq', \lambda']$ are well forked if*

$$\lambda(\mathscr{E}) \cap \lambda'(\mathscr{E}') \cap \mathscr{L}^? = \emptyset$$

we write $wf(r, r')$ when r and r' are well-forked and, for $\mathsf{G}, \mathsf{G}', wf(\mathsf{G}, \mathsf{G}')$ when $\llbracket \mathsf{G} \rrbracket \neq \bot \wedge \llbracket \mathsf{G}' \rrbracket \neq \bot \wedge \forall r \in \llbracket \mathsf{G} \rrbracket, r' \in \llbracket \mathsf{G}' \rrbracket : wf(r, r')$.

For a pomset $r = [\mathscr{E}, \leq, \lambda]$, let $\min r = \{e \in \mathscr{E} \mid \nexists e' \in \mathscr{E} : e' \neq e \wedge e' \leq e\}$ and, for a participant A, let $r_{\downarrow A} = [\mathscr{E}_{r,A}, \leq \cap (\mathscr{E}_{r,A}, \mathscr{E}_{r,A}), \lambda \mid_{\mathscr{E}_{r,A}}]$ be the pomset projected on A in r, where $\mathscr{E}_{r,A} = \{e \in \mathscr{E} \mid sbj(e) = A\}$, $\lambda \mid_{\mathscr{E}_{r,A}}$ denotes the restriction of the function λ to the subset $\mathscr{E}_{r,A}$ of its domain. The notion of well-branchedness requires the definition of *active* and *passive* participants of branches of g-choreographies.

Definition 2.8. *Given a branching g-choreography* $G + G'$*, let* $div_A(G, G') = (\tilde{l}_1, \tilde{l}_2)$ *with* $\tilde{l}_1, \tilde{l}_2 \subseteq \mathscr{L}$ *defined as*

$$\tilde{l}_1 = \bigcup_{r \in [\![G]\!]} \lambda \mid_{\mathscr{E}_{r,A}} (\min r_{\downarrow A}) \qquad and \qquad \tilde{l}_2 = \bigcup_{r' \in [\![G']\!]} \lambda \mid_{\mathscr{E}_{r',A}} (\min r'_{\downarrow A})$$

A participant $A \in \mathscr{P}$ *is*

- active in $G + G'$ *if* $\tilde{l}_1 \cup \tilde{l}_2 \subseteq \mathscr{L}^!$ $\wedge$ $\tilde{l}_1 \cap \tilde{l}_2 = \emptyset$ $\wedge$ $\tilde{l}_1 \neq \emptyset$ $\wedge$ $\tilde{l}_2 \neq \emptyset$
- passive in $G + G'$ *if* $\tilde{l}_1 \cup \tilde{l}_2 \subseteq \mathscr{L}^?$ $\wedge$ $\tilde{l}_1 \cap \tilde{l}_2 = \emptyset$ $\wedge$ $\tilde{l}_1 \neq \emptyset$ $\wedge$ $\tilde{l}_2 \neq \emptyset$.

An active (resp. passive) participant A in $G + G'$ must send (resp. receive) different messages to (resp. from) other participants at the branching starting points of g-choreographies. For instance, A is an active participant, while B and C are both passive participants in the branching g-choreography G in Figure 5.

Definition 2.9 (well-branchedness)**.** *A g-choreography* $G' + G''$ *is well-branched if*

1. *there is at most one active participant in* $G + G'$*,*

2. *all the other participants of* $G + G'$ *are passive participants.*

We write $wb(G, G')$ *when* $G' + G''$ *is well-branched.*

We define a variant of semantics of g-choreographies based on considering only well-branchedness.

Definition 2.10 (Semantics of g-choreographies)**.** *The semantics of a g-choreography is a family of pomsets defined as*

$$[\![0]\!] = \{\varepsilon\}$$

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

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

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

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

Interface automata   Interface automata [3, 4] play an important role in component-based design and verification. They capture input/output behaviours while the interfaces specify the possible interactions with the environment.

Definition 2.11 (Interface automata)**.** *An interface automaton* $M = (V, v_0, \mathscr{A}, \mathscr{T})$ *is a 4-tuple, where*

1. V *is a finite set of states,*

2. $v_0 \in V$ *is the initial state,*

3. $\mathscr{A} = \mathscr{A}^I \cup \mathscr{A}^O \cup \mathscr{A}^H$ *is the set of actions, where $\mathscr{A}^I$, $\mathscr{A}^O$ and $\mathscr{A}^H$ are pair-wise disjoint sets of input, output and internal actions, respectively,*

4. $\mathscr{T} \subseteq V \times \mathscr{A} \times V$ *is a set of transitions.*

We write $v \xrightarrow{a} v'$ instead of $(v, a, v') \in \mathscr{T}$ when the set of transitions $\mathscr{T}$ is clear from the context. Interface automata are able to synchronously interact with each other according to Definition 2.12.

Definition 2.12 (Composability and Product). *Two interface automata M and N are composable if*

$$\mathscr{A}_M^H \cap \mathscr{A}_N = \emptyset, \quad \mathscr{A}_M^I \cap \mathscr{A}_N^I = \emptyset, \quad \mathscr{A}_M^O \cap \mathscr{A}_N^O = \emptyset, \quad \mathscr{A}_N^H \cap \mathscr{A}_M = \emptyset.$$

We define $shared(M, N) = \mathscr{A}_M \cap \mathscr{A}_N$. If two interface automata M and N are composable, then we have $shared(M, N) = (\mathscr{A}_M^I \cap \mathscr{A}_N^O) \cup (\mathscr{A}_M^O \cap \mathscr{A}_N^I)$. If two interface automata M and N are composable, their synchronous product $M \otimes N$ is the interface automaton $M \otimes N = (V, v_0, \mathscr{A}, \mathscr{T})$,

$$
\begin{aligned}
V &= V_M \times V_N, \\
v_0 &= (v_{0M}, v_{0N}), \\
\mathscr{A} &= \mathscr{A}^I \cup \mathscr{A}^O \cup \mathscr{A}^H, \\
\mathscr{A}^I &= (\mathscr{A}_M^I \cup \mathscr{A}_N^I) \setminus shared(M,N), \\
\mathscr{A}^O &= (\mathscr{A}_M^O \cup \mathscr{A}_N^O) \setminus shared(M,N), \\
\mathscr{A}^H &= \mathscr{A}_M^H \cup \mathscr{A}_N^H \cup shared(M,N), \\
\mathscr{T} &= \{(v,u) \xrightarrow{a} (v',u) \mid (v \xrightarrow{a} v') \in \mathscr{T}_M \wedge a \notin shared(M,N) \wedge u \in V_N\} \\
&\quad \cup \{(v,u) \xrightarrow{a} (v,u') \mid (u \xrightarrow{a} u') \in \mathscr{T}_N \wedge a \notin shared(M,N) \wedge v \in V_M\} \\
&\quad \cup \{(v,u) \xrightarrow{a} (v',u') \mid (v \xrightarrow{a} v') \in \mathscr{T}_M \wedge (u \xrightarrow{a} u') \in \mathscr{T}_N \wedge a \in shared(M,N)\}
\end{aligned}
$$

Interactions of interface automata may lead to error states, which correspond to potential deadlocks.

Definition 2.13 (Error states). *The set of error states of the product of two interface automata M and N is defined by*

$$
\begin{aligned}
Error(M,N) = \{ (v,u) \in V_M \times V_N \mid \\
(\exists a \in shared(M,N) \wedge a \in \mathscr{A}_M^O : (v \xrightarrow{a} v') \in \mathscr{T}_M \wedge a \in \mathscr{A}_N^I : (u \xrightarrow{a} u') \notin \mathscr{T}_N) \vee \\
(\exists a \in shared(M,N) \wedge a \in \mathscr{A}_N^O : (u \xrightarrow{a} u') \in \mathscr{T}_N \wedge a \in \mathscr{A}_M^I : (v \xrightarrow{a} v') \notin \mathscr{T}_M) \}
\end{aligned}
$$

3 Group Interface Automata

We propose an extension of interface automata called group interface automata (GIA) as a convenient representation of local views of choreographies.

Definition 3.1 (Group interface automata). *A group interface automaton $\mathscr{I} = (V, v_0, \mathscr{G}, \mathscr{A}, \mathscr{T})$ 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. *$\mathscr{G} \subseteq \mathscr{P}$ is a finite set of participants,*

4. $\mathscr{A} = \mathscr{A}^I \cup \mathscr{A}^O \cup \mathscr{A}^H$ *is a set of interfaces (actions), where*

$$\mathscr{A}^I = (\mathscr{P} \setminus \mathscr{G}) \times \mathscr{G} \times \{?\} \times \mathscr{M}, \quad \mathscr{A}^O = \mathscr{G} \times (\mathscr{P} \setminus \mathscr{G}) \times \{!\} \times \mathscr{M}, \quad \mathscr{A}^H = \mathscr{G} \times \mathscr{G} \times \{!?\} \times \mathscr{M},$$

5. $\mathscr{T} \subseteq V \times \mathscr{A} \cup \{\tau\} \times V$ *is a set of transitions.*

The symbols "?", "!", "!?" mark input, output and internal actions respectively. Action τ represents a special internal computation. Given a GIA $\mathscr{I}$, we write $\mathscr{I}_{\mathscr{G}}$ when we want to highlight the group of participants in $\mathscr{I}$ and also $\mathscr{I}_A$ instead of $\mathscr{I}_{\{A\}}$. $AB?m$ and $AB!m$ and $AB!?m$ abbreviate $(A, B, ?, m) \in \mathscr{A}^I$, $(A, B, !, m) \in \mathscr{A}^O$, $(A, B, !?, m) \in \mathscr{A}^H$, respectively. The subject and object of transition actions are

$$sbj(AB!m) = obj(AB?m) = sbj(AB!?m) = A, \qquad obj(AB!m) = sbj(AB?m) = obj(AB!?m) = B$$

Figure 3 shows some instances of GIA. Let us comment on $\mathscr{I}_A$ and $\mathscr{I}_C$. The rectangles specify the scope of the machines, the arrow labeled *ACm* leaving from the border of $\mathscr{I}_A$ and the arrow labeled *ACm* leading to the border of $\mathscr{I}_C$ represent the output and input interfaces of $\mathscr{I}_A$ and $\mathscr{I}_C$ respectively.

Definition 3.2 (Shared interface). *Given two GIA $\mathscr{I} = (V, v_0, \mathscr{G}, \mathscr{A}, \mathscr{T})$ and $\mathscr{I}' = (V', v'_0, \mathscr{G}', \mathscr{A}', \mathscr{T}')$, the shared inputs, shared outputs and shared internals between $\mathscr{I}$ and $\mathscr{I}'$ are $si(\mathscr{I}, \mathscr{I}')$, $so(\mathscr{I}, \mathscr{I}')$, $sh(\mathscr{I}, \mathscr{I}')$ respectively, defined as*

$$si(\mathscr{I}, \mathscr{I}') = \{AB?m \in \mathscr{A} \mid AB!m \in \mathscr{A}'\} \cup \{AB?m \in \mathscr{A}' \mid AB!m \in \mathscr{A}\},$$
$$so(\mathscr{I}, \mathscr{I}') = \{AB!m \in \mathscr{A} \mid AB?m \in \mathscr{A}'\} \cup \{AB!m \in \mathscr{A}' \mid AB?m \in \mathscr{A}\},$$
$$sh(\mathscr{I}, \mathscr{I}') = \{AB!?m \mid AB!m \in \mathscr{A} \wedge AB?m \in \mathscr{A}'\} \cup \{AB!?m \mid AB!m \in \mathscr{A}' \wedge AB?m \in \mathscr{A}\}.$$

The composition of GIA assumes that each pair of automata $\mathscr{I}$ and $\mathscr{I}'$ uses a channel from $\mathscr{I}$ to $\mathscr{I}'$ (resp. $\mathscr{I}'$ to $\mathscr{I}$) to send messages from $\mathscr{I}$ to $\mathscr{I}'$ (resp. $\mathscr{I}'$ to $\mathscr{I}$), In CFSMs [2] 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. For our purposes, it is enough to assume that the size of the buffers of each channel is one. As describe below, our definition of composition induces one-size buffer channels between each two participants, When an automata 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 buffer first, then consumes the message if there is any.

Definition 3.3 ($\otimes$-product). *Two GIA $\mathscr{I}' = (V', v'_0, \mathscr{G}', \mathscr{A}', \mathscr{T}')$ and $\mathscr{I}'' = (V'', v''_0, \mathscr{G}'', \mathscr{A}'', \mathscr{T}'')$ are composable if $\mathscr{G}' \cap \mathscr{G}'' = \emptyset$. If two GIA $\mathscr{I}'$ and $\mathscr{I}''$ are composable, $\mathscr{I}' \otimes \mathscr{I}''$ is the GIA $\mathscr{I} = (V' \times V'', (v'_0, v''_0), \mathscr{G}' \cup \mathscr{G}'', \mathscr{A}^I \cup \mathscr{A}^O \cup \mathscr{A}^H, \mathscr{T})$, where*

$$\mathscr{A}^I = (\mathscr{A}'^I \cup \mathscr{A}''^I) \setminus si(\mathscr{I}', \mathscr{I}''),$$
$$\mathscr{A}^O = (\mathscr{A}'^O \cup \mathscr{A}''^O) \setminus so(\mathscr{I}', \mathscr{I}''),$$
$$\mathscr{A}^H = \mathscr{A}'^H \cup \mathscr{A}''^H \cup sh(\mathscr{I}', \mathscr{I}''),$$
$$\mathscr{T} = \{(v', v'') \xrightarrow{\alpha} (u', v'') \mid (v' \xrightarrow{\alpha} u') \in \mathscr{T}' \wedge \alpha \notin (si(\mathscr{I}', \mathscr{I}'') \cup so(\mathscr{I}', \mathscr{I}'')) \wedge v'' \in V''\}$$
$$\cup \{(v', v'') \xrightarrow{\alpha} (v', u'') \mid (v'' \xrightarrow{\alpha} u'') \in \mathscr{T}'' \wedge \alpha \notin (si(\mathscr{I}', \mathscr{I}'') \cup so(\mathscr{I}', \mathscr{I}'')) \wedge v' \in V'\}$$
$$\cup \{(v', v'') \xrightarrow{AB!?m} (u', u'') \mid (v' \xrightarrow{AB!m} u') \in \mathscr{T}' \wedge (v'' \xrightarrow{AB?m} u'') \in \mathscr{T}''\}$$
$$\cup \{(v', v'') \xrightarrow{AB!?m} (u', u'') \mid (v'' \xrightarrow{AB!m} u'') \in \mathscr{T}'' \wedge (v' \xrightarrow{AB?m} u') \in \mathscr{T}'\}.$$
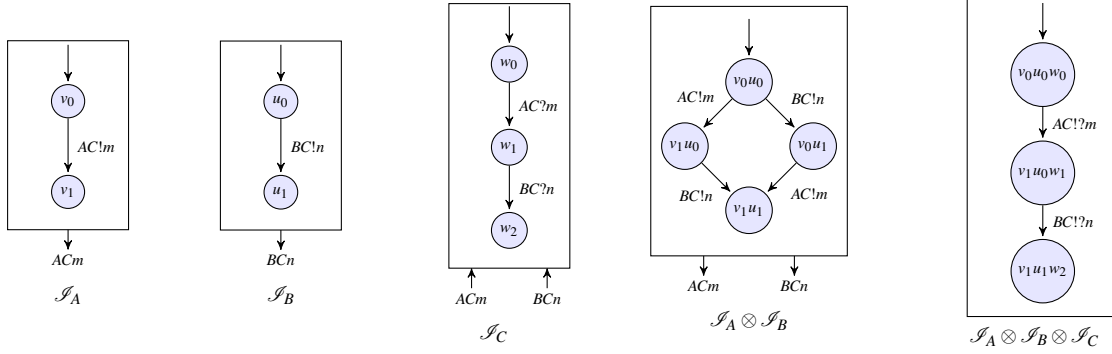
Figure 3: The ⊗-product among three composable GIA

Interactions among GIA are captured by their product. Figure 3 shows the ⊗-products among three pair-wise composable GIA. The product $\mathscr{I}_A \otimes \mathscr{I}_B$ yields the interaction between GIA $\mathscr{I}_A$ and $\mathscr{I}_B$. Due to the sets of interfaces of $\mathscr{I}_A$ and $\mathscr{I}_B$ being disjoint, then, according to Definition 3.3, the structure of $\mathscr{I}_A \otimes \mathscr{I}_B$ is interleaving the transitions of $\mathscr{I}_A$ and $\mathscr{I}_B$. $\mathscr{I}_A \otimes \mathscr{I}_B \otimes \mathscr{I}_C$ has only internal interfaces since the set of interfaces of $\mathscr{I}_A \otimes \mathscr{I}_B$ and $\mathscr{I}_C$ are complementary. Moreover, the operator $\_ \otimes \_$ is commutative and associative, which can be proved by showing the bisimulation of sets of states between two GIA. Notice that the composition of interface automata is "optimistic" in the sense that it allows one to make automata interact as long as they can execute some traces avoiding error states. This does not fit choreographic approaches like ours. In fact, the idea of choreographies is to specify systems that never run into error states. Hence, composition in GIA tries to single out all the error states emerging from interactions.

Theorem 3.4 (Commutativity and associativity)**.** *Let $\mathscr{I}$, $\mathscr{I}'$ and $\mathscr{I}''$ be three GIA, then $\mathscr{I} \otimes \mathscr{I}' = \mathscr{I}' \otimes \mathscr{I}$. If $\mathscr{I}$, $\mathscr{I}'$ and $\mathscr{I}''$ are pair-wise composable, then $(\mathscr{I} \otimes \mathscr{I}'') \otimes \mathscr{I}' = \mathscr{I} \otimes (\mathscr{I}' \otimes \mathscr{I}'')$.*

Next, similarly to error states in interface automata, we define error states in the ⊗-product of GIA. Let the dual of an action be defined as

$$dual(AB!m) = AB?m \qquad dual(AB?m) = AB!m \qquad dual(AB!?m) = AB!?m$$

and the dual of a string $\alpha_0 \alpha_1 \cdots \alpha_n$ as $dual(\alpha_0 \alpha_1 \cdots \alpha_n) = dual(\alpha_0)dual(\alpha_1)\cdots dual(\alpha_n)$. Given a transition $v \xrightarrow{\alpha} u$, we define

$$\#(v \xrightarrow{\alpha} u, A, B) = \begin{cases} \alpha & \text{if } sbj(\alpha) = A \text{ and } obj(\alpha) = B \\ \tau & \text{otherwise} \end{cases} \tag{1}$$

where we overload τ to represent the empty string. Then, we extend $\#(\_,\_,\_)$ to sequences of transitions $\mathsf{t} = v \xrightarrow{\alpha_0} \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} u$ as follows

$$\#(\mathsf{t}, A, B) = \#(\alpha_0, A, B)\#(\alpha_i, A, B)\cdots\#(\alpha_n, A, B).$$

Given a string ω, the prefixes of ω are $\mathsf{Pref}(\omega) = \{\mu | \exists v . \omega = \mu v\}$. Given a non-empty string $\omega\alpha$, we define an operation that removes the last action if it is an output, where

$$\mathsf{ro}(\omega\alpha) = \begin{cases} \omega & \text{if } \alpha \in \mathscr{L}^! \\ \omega\alpha & \text{otherwise} \end{cases} \tag{2}$$

Definition 3.5 (Error states of the $\otimes$-product). *Let v and v' be states respectively of GIAs $\mathscr{I}$ and $\mathscr{I}'$. We say that v has an unmatched shared output by v' if there exists $AB!m \in so(\mathscr{I}, \mathscr{I}')$ and a sequence of transitions $\mathsf{t} = v \xrightarrow{AB!m} \xrightarrow{\beta_0} \cdots \xrightarrow{\beta_k} u \in \mathscr{I}$ then for all $\mathsf{t}' = v' \xrightarrow{\alpha_0} \cdots \xrightarrow{\alpha_n} u' \in \mathscr{I}'$ either of the following conditions holds*

1. *$\alpha_i \neq AB?m$ 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 $0 \leq i < \hat{h}$ for which $\alpha_i \in si(\mathscr{I}, \mathscr{I}') \cup so(\mathscr{I}, \mathscr{I}')$ with $sbj(\alpha_i) = B$ and*

$$obj(\alpha_i) = C \neq A \implies ro(\#(\mathsf{t}'_{\leq \hat{h}}, B, C)) \notin Pref(dual(\#(\mathsf{t}, C, B)))$$

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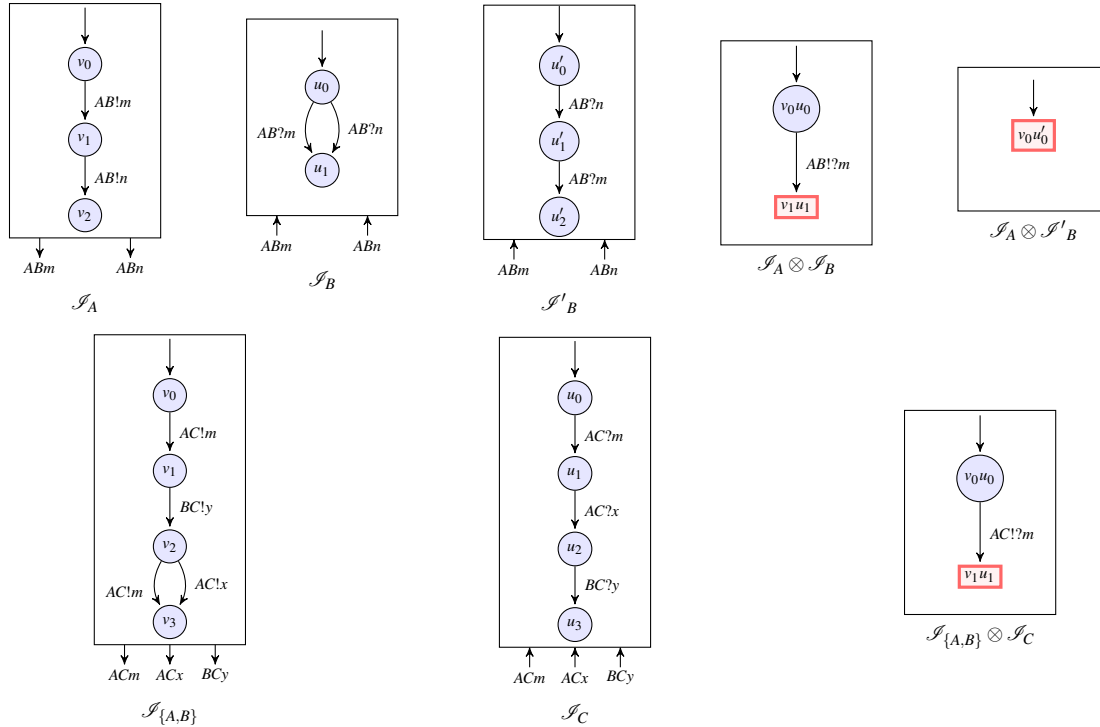

Figure 4: Error states in GIA

The definition of error states detects the situation where at least one shared output is not consumed. Figure 4 shows three classes of error states represented by red boxes. (1) The state $v_1 u_1$ of $\mathscr{I}_A \otimes \mathscr{I}_B$ is an error state (by (1) in Definition 3.5) since the shared output $AB!n$ in state v_1 does not have a corresponding input $AB?m$ in state u_1. In other words, when $\mathscr{I}_A$ wants to send n to $\mathscr{I}_B$ at state v_1, $\mathscr{I}_B$ will never receive n at state u_1. (2) The initial state $v_0 u'_0$ of $\mathscr{I}_A \otimes \mathscr{I}'_B$ is an error state (by (2) in Definition 3.5): The shared output $AB!m$ departing from v_0 in $\mathscr{I}_A$ has a corresponding input $AB?m$ in the sequence from u'_0 in $\mathscr{I}'_B$, however, the first transition from u'_0 is $AB?n$ which is another shared interface between $\mathscr{I}_A$ and $\mathscr{I}'_B$. In other words, $\mathscr{I}_A$ wants to send m to $\mathscr{I}'_B$ at state v_0, but $\mathscr{I}'_B$ 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 awayfrom n. (3) The state $v_1 u_1$ of $\mathscr{I}_{\{A,B\}} \otimes \mathscr{I}_C$ is an error state (by (2) in Definition 3.5). The shared output $BC!y$ in v_1 has the corresponding input $BC?y$ in u_1, and the first transition from u_1 is $AC?x$, which is another shared interface between $\mathscr{I}_{\{A,B\}}$ and $\mathscr{I}_C$. However, the corresponding shared output $AC!x$ is not in each sequence from state v_2 in $\mathscr{I}_{\{A,B\}}$. In other words, $\mathscr{I}_{\{A,B\}}$ sends y to $\mathscr{I}_C$ in v_1, then sends m to $\mathscr{I}_C$ in v_2, but y is not consumed by $\mathscr{I}_C$ since $\mathscr{I}_C$ wants to first receive x from $\mathscr{I}_{\{A,B\}}$ in u_1, however, x is not sent in v_2 in $\mathscr{I}_{\{A,B\}}$.

Theorem 3.6. *Given two composable GIA $\mathscr{I}$ and $\mathscr{I}'$, an error state (v, v') exists in $\mathscr{I} \otimes \mathscr{I}'$ if and only if at least a send action in the shared output departing from v or v' is never consumed.*

4   Group Interface Automata Based Verification

We advocate GIA for the verification of well-formedness of g-choreographies. According to [7], well-formedness of g-choreographies implies deadlock freedom. Now, we show that the well-formedness of g-choreographies is equivalent to the non-existence of error states and parallel, branching error states in the corresponding GIA. Therefore, non-existence of (parallel, branching) error states implies deadlock freedom. The following diagram describes our approach.

1. Projections on participants A 2. Remove all *removable* τ-transitions $\overline{G_{\downarrow A}}$

Global Choreography G   Group Interface Automata $G_{\downarrow A}$

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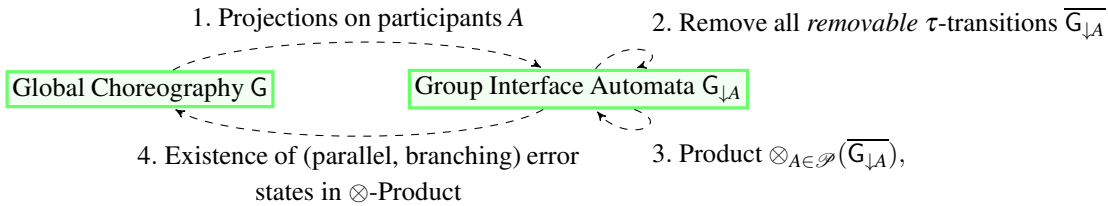
Roughly speaking, (1) projection yields a set of GIA corresponding to each local participant involved in the g-choreography; (2) projection introduces τ-transitions, that is internal transitions that do not represent communications; some of those τ-transitions may lead to spurious error states in the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

4.1   Projection

We define some auxiliary notions before introducing the projection operation.

1. Let $\mathscr{I}' = (V', v_0', \mathscr{G}', \mathscr{A}', \mathscr{T}')$ and $\mathscr{I}'' = (V'', v_0'', \mathscr{G}'', \mathscr{A}'', \mathscr{T}'')$ be two $\mathscr{G}$-interface automata. We define $\mathscr{I}' \times \mathscr{I}'' = (V' \times V'', (v_0', v_0''), \mathscr{G}' \cup \mathscr{G}'', \mathscr{A}' \cup \mathscr{A}'', \mathscr{T})$, where $((v', v''), \alpha, (u', u'')) \in \mathscr{T}$ iff

$$((v', \alpha, u') \in \mathscr{T}' \text{ and } v'' = u'' \in V'') \quad \text{or} \quad ((v'', \alpha, u'') \in \mathscr{T}'' \text{ and } v' = u' \in V').$$

2. Let $\{u/v\}\mathscr{I}$ be the automaton obtained by substituting the state v with the state u.

3. Let $\mathscr{I} \odot \mathbf{n}$ be the automaton $(V \times \mathbf{n}, (v_0, n), \mathscr{G}, \mathscr{A}, \mathscr{T} \odot \mathbf{n})$, where $\mathscr{T} \odot \mathbf{n} = \{((v, n), \alpha, (u, n)) \mid (v, \alpha, u) \in \mathscr{T}\}$.

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

Then, we define a projection function by induction on the syntax of G returning a triple $(\mathscr{I}_A, v_0, v_e)$, where $\mathscr{I}_A$ is a GIA, v_0 is its initial state, and v_e is the special state of $\mathscr{I}$ 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 $(\mathscr{I}_A, v_0, v_e) \odot \mathbf{n}$ to represent $(\mathscr{I}_A \odot \mathbf{n}, (v_0, n), (v_e, n))$.

Definition 4.1 (Projection from g-choreographies to GIA). *Let* G *be a g-choreography. The projections* $G_{\downarrow A}$ *are defined as follows.*

$$
G_{\downarrow A} = \begin{cases}
\begin{array}{l} \rightarrow \boxed{\xrightarrow{} (v_0) \xrightarrow{\tau} (v_e)} \end{array} & \text{if } G = 0 \\[2ex]
\begin{array}{l} \rightarrow \boxed{\xrightarrow{} (v_0) \xrightarrow{\tau} (v_e)} \end{array} & \text{if } G = B \xrightarrow{m} C \\[2ex]
\begin{array}{l} \rightarrow \boxed{\xrightarrow{} (v_0) \xrightarrow{AB!m} (v_e)} \rightarrow ABm \end{array} & \text{if } G = A \xrightarrow{m} B \text{ and } v_0 \neq v_e \\[2ex]
\begin{array}{l} \rightarrow \boxed{\xrightarrow{} (v_0) \xrightarrow{BA?m} (v_e)} \leftarrow BAm \end{array} & \text{if } G = B \xrightarrow{m} A \text{ and } v_0 \neq v_e \\[2ex]
(\mathscr{I}'_A \circ \{v_e/u_0\} \mathscr{I}''_A, v_0, u_e) & \begin{array}{l} \text{if } G = G'; G'' \\ \quad \text{and } (\mathscr{I}'_A, v_0, v_e) = G'_{\downarrow A} \odot \mathbf{1} \\ \quad \text{and } (\mathscr{I}''_A, u_0, u_e) = G''_{\downarrow A} \odot \mathbf{2} \end{array} \\[3ex]
(\{u_e/v_e\} \mathscr{I}'_A \circ \{v_0/u_0\} \mathscr{I}''_A, v_0, u_e) & \begin{array}{l} \text{if } G = G' + G'' \\ \quad \text{and } (\mathscr{I}'_A, v_0, v_e) = G'_{\downarrow A} \odot \mathbf{1} \\ \quad \text{and } (\mathscr{I}''_A, u_0, u_e) = G''_{\downarrow A} \odot \mathbf{2} \end{array} \\[3ex]
(\mathscr{I}'_A \times \mathscr{I}''_A, (v_0, u_0), (v_e, u_e)) & \begin{array}{l} \text{if } G = G' \mid G'' \\ \quad \text{and } (\mathscr{I}'_A, v_0, v_e,) = \overline{G'_{\downarrow A}} \odot \mathbf{1} \\ \quad \text{and } (\mathscr{I}''_A, u_0, u_e) = \overline{G''_{\downarrow A}} \odot \mathbf{2} \end{array}
\end{cases}
\tag{3}
$$

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 τ 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, τ) if the projection is on the sender (resp. receiver, other participants). (3) The projection $(G'; G'')_{\downarrow A}$ is obtained 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 $(G' \mid G'')_{\downarrow A}$ is generated by interleaving the transitions of $\overline{G'_{\downarrow A}}$ and $\overline{G''_{\downarrow A}}$, where $\overline{G'_{\downarrow A}}$ and $\overline{G''_{\downarrow A}}$ are the GIA without removable τ-transitions refined from $G'_{\downarrow A}$ and $G''_{\downarrow A}$ respectively. Removable τ-transitions will be introduced in next subsection.

4.2   Removability of τ-transitions

Projection introduces τ-transitions which induce error states in the $\otimes$-product. However, some error states may be spurious, that is, they do not correspond to deadlocks. For instance, Figure 5 shows a

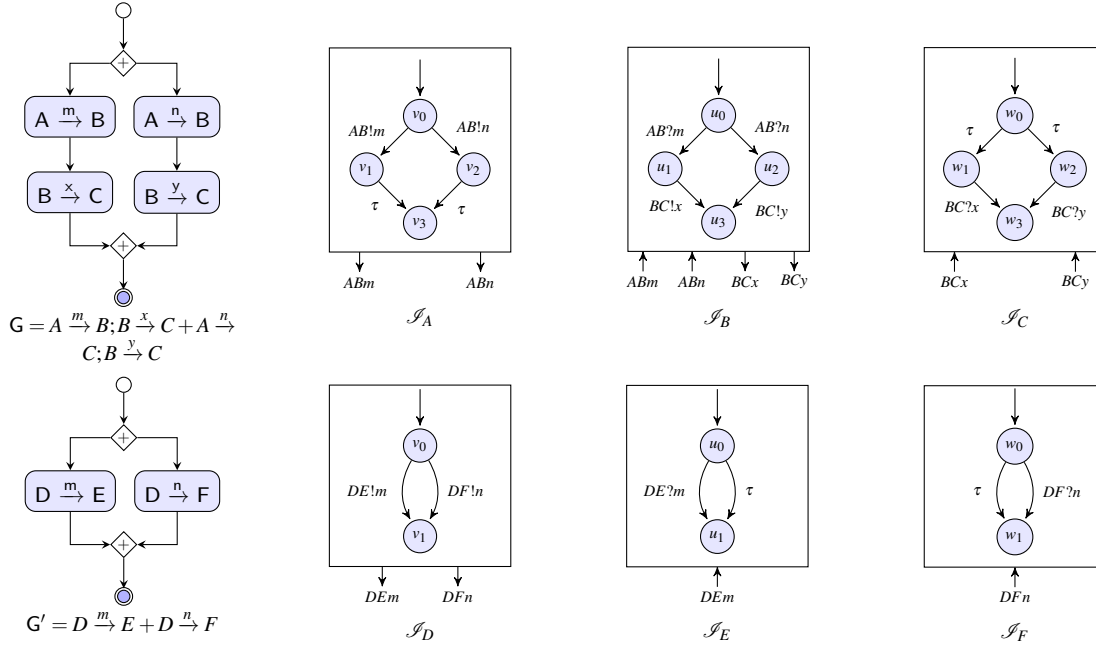well-formed g-choreography G, a not well-formed g-choreography G' and their projections.

Figure 5: Spurious error states

The states $v_3u_1w_2$ and $v_3u_2w_1$ in $\mathscr{I}_A \otimes \mathscr{I}_B \otimes \mathscr{I}_C$ and $v_0u_1w_1$, $v_0u_0w_1$, $v_0u_1w_0$ in $\mathscr{I}_D \otimes \mathscr{I}_E \otimes \mathscr{I}_F$ are error states according to Definition 3.5. However, the error states in $\mathscr{I}_A \otimes \mathscr{I}_B \otimes \mathscr{I}_C$ are spurious error states, whilst the error states in $\mathscr{I}_D \otimes \mathscr{I}_E \otimes \mathscr{I}_F$ do reflect deadlocks in G'. Therefore, in order to disingiush spurious error states from error states, some τ-transitions will be accounted for as removable. We first define the language between two states in GIA. We use α, β to range over letters and ω to range over words; as usual, the concatenation of words ω and ω' is written as $\omega\omega'$ and τ is the neutral element of word concatenation. Given a GIA $\mathscr{I} = (V, v_0, \mathscr{G}, \mathscr{A}, \mathscr{T})$, the language between two states $v, v' \in V$ is

$$\mathscr{L}(v,v') = \begin{cases} \{\alpha_0 \cdots \alpha_n \mid v \xrightarrow{\alpha_0} v_1 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} v'\} & if\ v \neq v' \\ \{\alpha_0 \cdots \alpha_n \mid v \xrightarrow{\alpha_0} v_1 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} v'\} \cup \{\tau\} & if\ v = v'. \end{cases}$$

Next, we give the definition of removability of τ-transitions.

Definition 4.2 (Removable τ-transitions). *Given a GIA $\mathscr{I} = (V, v_0, \mathscr{G}, \mathscr{A}, \mathscr{T})$, a τ-transition $v \xrightarrow{\tau} v_\tau \in \mathscr{T}$ is removable if, for all $v' \in V$, $\mathscr{L}(v,v') \neq \emptyset$ and for all $\alpha\omega \in \mathscr{L}(v,v')$ we have*

$$\alpha\omega \neq \tau \implies \exists v'_\tau \in V : \mathscr{L}(v_\tau, v'_\tau) = \{\tau\} \wedge v'\ \alpha\omega\text{-compatible to } v'_\tau$$

where v' is $\alpha\omega$-compatible to v'_τ if for all $\bar{v} \in V$ such that $\mathscr{L}(v'_\tau, \bar{v}) \neq \{\tau\} \wedge \mathscr{L}(v'_\tau, \bar{v}) \neq \emptyset$

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

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

$$\mathscr{L}(v'_\tau, \bar{v})\mathscr{L}(\bar{v}, \hat{v}) = \{\alpha\omega\} \qquad and \qquad \mathscr{L}(\hat{v}, v') \in \{\emptyset, \{\tau\}\} \ni \mathscr{L}(v', \hat{v})$$

Roughly speaking, a τ-transition $v \xrightarrow{\tau} v_\tau$ is removable when either of the two following conditions are satisfied. (1) The τ-transition $v \xrightarrow{\tau} v_\tau$ does not affect the non-deterministic choices from the state v. (2) The first non τ-transition actions leaving from states v and v_τ are all input interfaces or all output interfaces.

For instance, Figure 6 shows three simple instances of the removability of τ-transitions, the first two machines reflect case (2) and case (1) respectively. In the GIA $\mathscr{I}_A$, the τ-transitions $v \xrightarrow{\tau} v_\tau$ and $v \xrightarrow{\tau} v'_\tau$ denoted by red colour are removable. In the GIA $\mathscr{I}_B$, $v_\beta \xrightarrow{\tau} v'_\tau$ is removable, and if α, β are both inputs or both outputs, then the τ-transition $v \xrightarrow{\tau} v_\tau$ is removable (and otherwise it is not). In the GIA $\mathscr{I}_C$, $v \xrightarrow{\tau} v_\tau$ is not removable since it affects the choices from v.
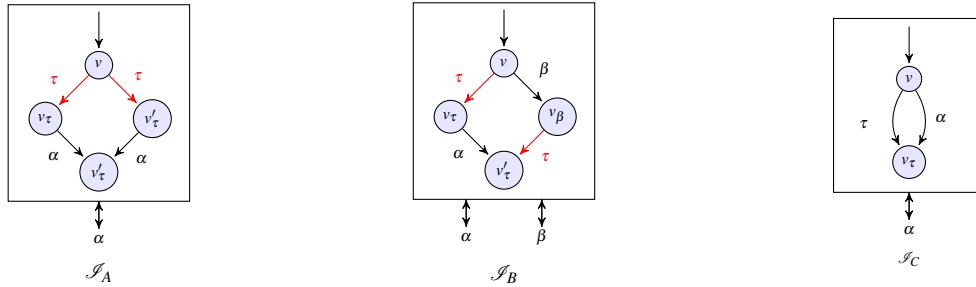

Figure 6: Removability of τ-transitions

After finding all removable τ-transitions in projections, we construct the GIA without removable τ-transitions from τ-equivalence classes.

Definition 4.3 (τ-Equivalence class of states)**.** *Let $\simeq_\tau$ be the smallest equivalence relation containing all $v \simeq_\tau v'$, where $v \xrightarrow{\tau} \cdots \xrightarrow{\tau} v'$ or $v \xrightarrow{\tau} \cdots \xrightarrow{\tau} v$, and each τ-transition is removable. We write $[v]_{\simeq_\tau}$ as the equivalence class of the state v w.r.t. $\simeq_\tau$.*

We define $[V]_{\simeq_\tau} = \bigcup_{v \in V} [v]_{\simeq_\tau}$, where V is a set of states.

Definition 4.4. *Let $\mathscr{I} = (V, v_0, \mathscr{G}, \mathscr{A}, \mathscr{T})$ be a GIA. Then $\overline{\mathscr{I}} = (\overline{V}, \overline{v_0}, \overline{\mathscr{G}}, \overline{\mathscr{A}}, \overline{\mathscr{T}})$ is the GIA without removable τ-transitions constructed from $\mathscr{I}$, where*

$$\overline{V} = \{[v]_{\simeq_\tau} \mid v \in V\} \cup \{\emptyset\}, \quad \overline{v_0} = [v_0]_{\simeq_\tau}, \quad \overline{\mathscr{G}} = \mathscr{G}, \quad \overline{\mathscr{A}} = \mathscr{A}, \quad \overline{\mathscr{T}} = \{([v]_{\simeq_\tau}, \alpha, [v']_{\simeq_\tau}) \mid (v, \alpha, v') \in \mathscr{T}\}.$$

Theorem 4.5. *Let G be a g-choreography, $G_{\downarrow A} = \mathscr{I}_A = (V, v_0, \{A\}, \mathscr{A}, \mathscr{T})$ be a GIA projected from participant A where $A \in \mathscr{P}$. Let $\overline{G_{\downarrow A}} = \overline{\mathscr{I}_A} = (\overline{V}, \overline{v_0}, \{A\}, \overline{\mathscr{A}}, \overline{\mathscr{T}})$ be the GIA without removable τ-transitions constructed from $G_{\downarrow A}$. Then $G_{\downarrow A}$ and $\overline{G_{\downarrow A}}$ are language equivalent.*

For instance, Figure 7 shows the GIA with without removable τ-transitions $\overline{\mathscr{I}_A}$, $\overline{\mathscr{I}_B}$ and $\overline{\mathscr{I}_C}$ refined from G in Figure 5, as well as their $\otimes$-product $\overline{\mathscr{I}_A} \otimes \overline{\mathscr{I}_B} \otimes \overline{\mathscr{I}_C}$ which does not have any error states.

4.3   Parallel and branching error state

In GIA based verification, we need to supplement error states with new concepts called parallel error states and branching error states. As we have seen, error states capture deadlocks. However, the non-existence of error states is not sufficient to guarantee the well-formedness of parallel and branching compositions. This leads to the definition of parallel and branching error states. Again, we need some
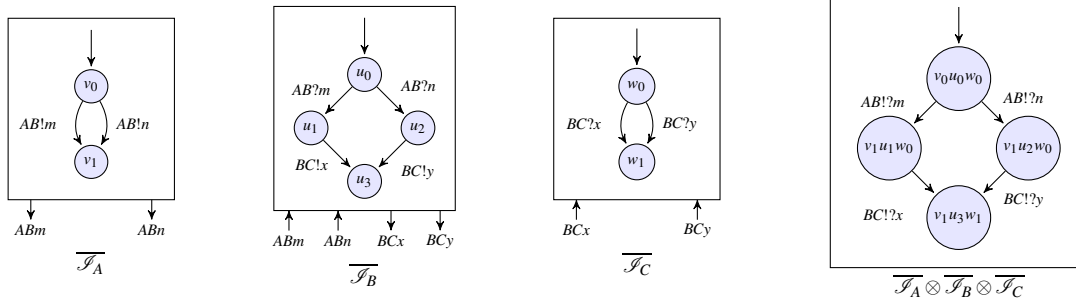
Figure 7: Refined GIA and their $\otimes$-product

auxiliary notation first. We extend the definition of *sbj* and *obj* from Def.2.2 to sequences in the obvious way

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

also, we set $sobj(AB!?m) = \{A,B\}$ and define

$$sobj(\{\alpha_0\alpha_1\cdots\alpha_n\}) = sbj(\{\alpha_0\alpha_1\cdots\alpha_n\}) \cup obj(\{\alpha_0\alpha_1\cdots\alpha_n\}).$$

$$sobj(\{\omega \mid \omega = \alpha_0\alpha_1\cdots\alpha_n\}) = \bigcup_{0<i<n} sobj(\alpha_i)$$

Definition 4.6 (Parallel error states). *Given a parallel composition* $\mathsf{G} = \mathsf{G}' \mid \mathsf{G}''$, *state v is a parallel error state in* $\otimes_{A\in\mathscr{P}}\overline{(\mathsf{G}' \mid \mathsf{G}'')_{\downarrow A}} = (V, v_0, \mathscr{G}, \mathscr{A}, \mathscr{T})$ *if*

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

Let $parError(\otimes_{A\in\mathscr{P}}\overline{(\mathsf{G}' \mid \mathsf{G}'')_{\downarrow A}})$ *be the set of parallel error states of* $\otimes_{A\in\mathscr{P}}\overline{(\mathsf{G}' \mid \mathsf{G}'')_{\downarrow A}}$.

Parallel error states capture the non well-formedness of parellel g-choreographies that cannot be detected by error states. According to Definition 2.7, we know that well-forkedness needs there is no common interactions existing in two threads. And according to Definition 4.1, we know that the projection of parallel compositions is constructed by interleaving the transitions from two threads. Roughly, if the product generates a state v with transitions with the same label α that form a diamond, then v is a parallel error state. For instance, in Figure 8, the parallel composition G_0 is not well-forked. There are no error states in $\overline{\mathscr{I}_A} \otimes \overline{\mathscr{I}_B}$ (cf. Figure 8), but the initial state $v_0 u_0$ of $\overline{\mathscr{I}_A} \otimes \overline{\mathscr{I}_B}$ is a parallel error states.

Definition 4.7 (Branching error states). *Given a branching composition* $\mathsf{G} = \mathsf{G}' + \mathsf{G}''$, *we say states v and v' ($v \neq v'$) are branching error states in* $\otimes_{A\in\mathscr{P}}\overline{(\mathsf{G}' + \mathsf{G}'')_{\downarrow A}} = (V, v_0, \mathscr{G}, \mathscr{A}, \mathscr{T})$, *if either of the following conditions holds*

1. $v = v_0 \wedge |\mathscr{T}| = 1$,

2. $\mathscr{L}(v_0, v) = \{\tau\} \wedge (\exists(v \xrightarrow{AB!?m} u \wedge v \xrightarrow{CD!?n} w) : u \neq w \wedge (A \neq C \vee AB!?m = CD!?n))$,

3. $(\mathscr{L}(v_0, v) \neq \{\tau\} \vee \mathscr{L}(v_0, v') \neq \{\tau\}) \wedge (\exists(v \xrightarrow{AB!?m} u \wedge v' \xrightarrow{AB!?m} u') :$
$$(A \notin sobj(\mathscr{L}(v_0, v)) \wedge A \notin sobj(\mathscr{L}(v_0, v'))) \vee$$
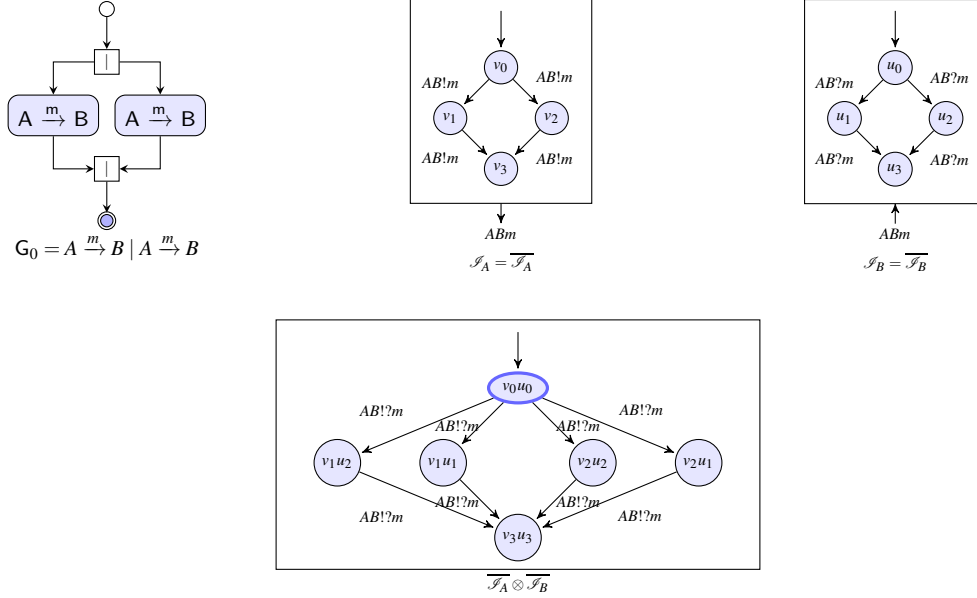$$(B \notin sobj(\mathscr{L}(v_0, v)) \wedge B \notin sobj(\mathscr{L}(v_0, v')))).$$

Figure 8: Parallel error state

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

Branching error states capture the non well-formedness of branching g-choreographies that can not be detected by error states. According to Definition 4.7, we summarise four classes of branching error states. Let G be a branching g-choreography.

(1) If the initial state of $\otimes_{A\in\mathscr{P}}(G'+G'')_{\downarrow A}$ just has only one transition, then the initial state is a branching error state. For instance, in Figure 9, we are able to see the initial state v_0u_0 denoted by a green diamond of $\overline{\mathscr{I}_A}\otimes\overline{\mathscr{I}_B}$ is a branching error state (by (1) in Definition 4.7) since there is only transitions from v_0u_0 to v_1u_1. (2) If the initial state of $\otimes_{A\in\mathscr{P}}(G'+G'')_{\downarrow A}$ has two transitions which have different subjects (sender) in transition actions, then, we define this initial state is a branching error state. For instance, in Figure 9, we are able to see the initial state v_0u_0 denoted by a green diamond of $\overline{\mathscr{I}_C}\otimes\overline{\mathscr{I}_D}$ is a branching error state (by (2) in Definition 4.7) 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. (3) If the initial state of $\otimes_{A\in\mathscr{P}}(G'+G'')_{\downarrow A}$ has two transitions which have the same transition actions, then, we define this initial state is a branching error state. For instance, in Figure 10, we are able to see the initial state $v_0u_0w_0$ denoted by a green diamond of $\overline{\mathscr{I}_A}\otimes\overline{\mathscr{I}_B}\otimes\overline{\mathscr{I}_C}$ is a branching error state (by (2) in Definition 4.7) since there are two transitions from $v_0u_0w_0$ to have the same actions $AB!?m$. (4) If there exist two states located in the execution paths from the initial state of $\otimes_{A\in\mathscr{P}}(G'+G'')_{\downarrow A}$, and when each of these two states has a transition which has the same transition action, while the subject of the action is not in the set of the subjects and objects of the strings from the initial states to these two states, then, we define these two states are branching error states. For instance, in Figure 10, we are able to see the states $v_1u_1w_0$ and $v_1u_2w_0$ denoted by green diamonds of $\overline{\mathscr{I}_A}\otimes\overline{\mathscr{I}_B}\otimes\overline{\mathscr{I}_C}$ are branching error states (by (3) in Definition 4.7) 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 $sobj(AB!?m)$.
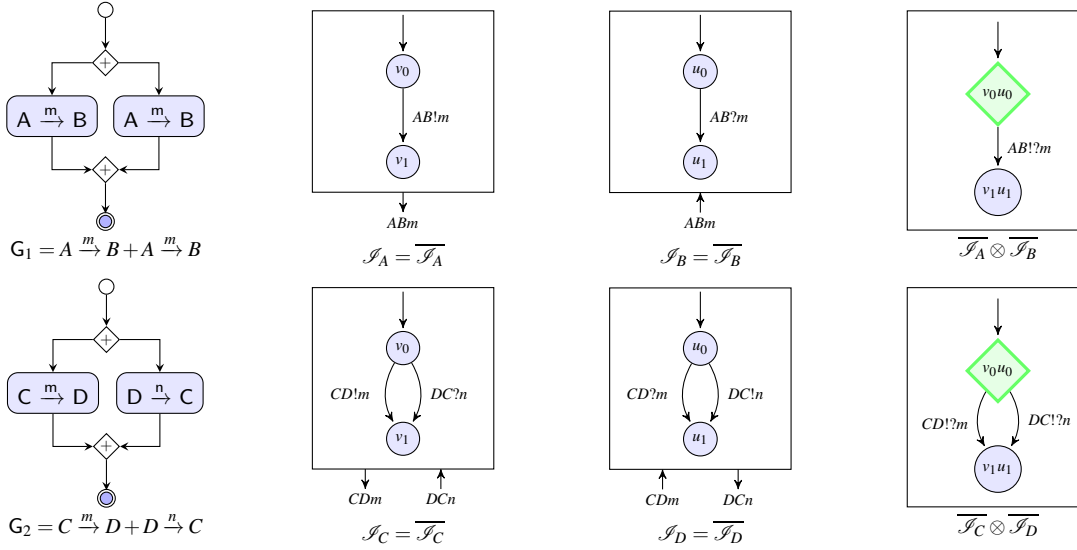
Figure 9: Branching error states in $\otimes_{A \in \mathscr{P}}(\mathsf{G}_{1 \downarrow A})$ and $\otimes_{C \in \mathscr{P}}(\mathsf{G}_{2 \downarrow C})$

4.4 Well-formedness and non-existence of (parallel, branching) error states

We have introduced projection from g-choreographies to GIA, the removability of τ-transitions, branching error states and parallel error states. Next, we write the main theorem below.

Theorem 4.8. *A g-choreography* G *is well-formed iff there are neither error states nor parallel, branching error states in* $\otimes_{A \in \mathscr{P}}(\overline{\mathsf{G}_{\downarrow A}})$, *where* $\overline{\mathsf{G}_{\downarrow A}}$ *is the GIA without removable τ-transition constructed from* $\mathsf{G}_{\downarrow A}$.

The theorem 4.8 shows that a g-choreography (Definition 2.1) is well-formed (Definition 2.10) if, and only if the $\otimes$-product (Definition 3.3) of the set of projections (Definition 4.1) without τ (Definition 4.2, 4.4) does not have error states (Definition 3.5), parallel error states(Definition 4.6) and branching error states (Definition 4.7).

5 Conclusion

We established a new way to check well-formedness based on an extension of interface automata. We adopted a variant of the semantics of g-choreograhies presented in [7]. Our semantics relaxes some conditions on well-formedness (more precisely the conditions for sequential, parallel, and branching compositions are stricter in [7] than here). In this paper we did not consider iterative g-choreograhies; however, extending our results to this case is conceptually straightforward as loops can be dealt with as done eg in [6]. The main limitation of our notion of well-formedness is in the treatment of well-branchedness that here we treat "syntactically". For instance, in G_1 of Figure 9, participants A and B are neither active nor passive participants according to our since A and B behave exactly same in both branches making the choice non well-branched. In fact, to identify active and passive participants our semantics relies on divergence points imposed by the syntax of g-choreograhies instead of using the (semantic) concept of prefix-maps. This latter concept allows one to identify, for each participant, where (if at all) the participant becomes aware of the choice. Hence, using prefix maps the choice G_1 of Figure 9 is well-branched since both A and B behaves uniformly in the branches (A and B are both passive and
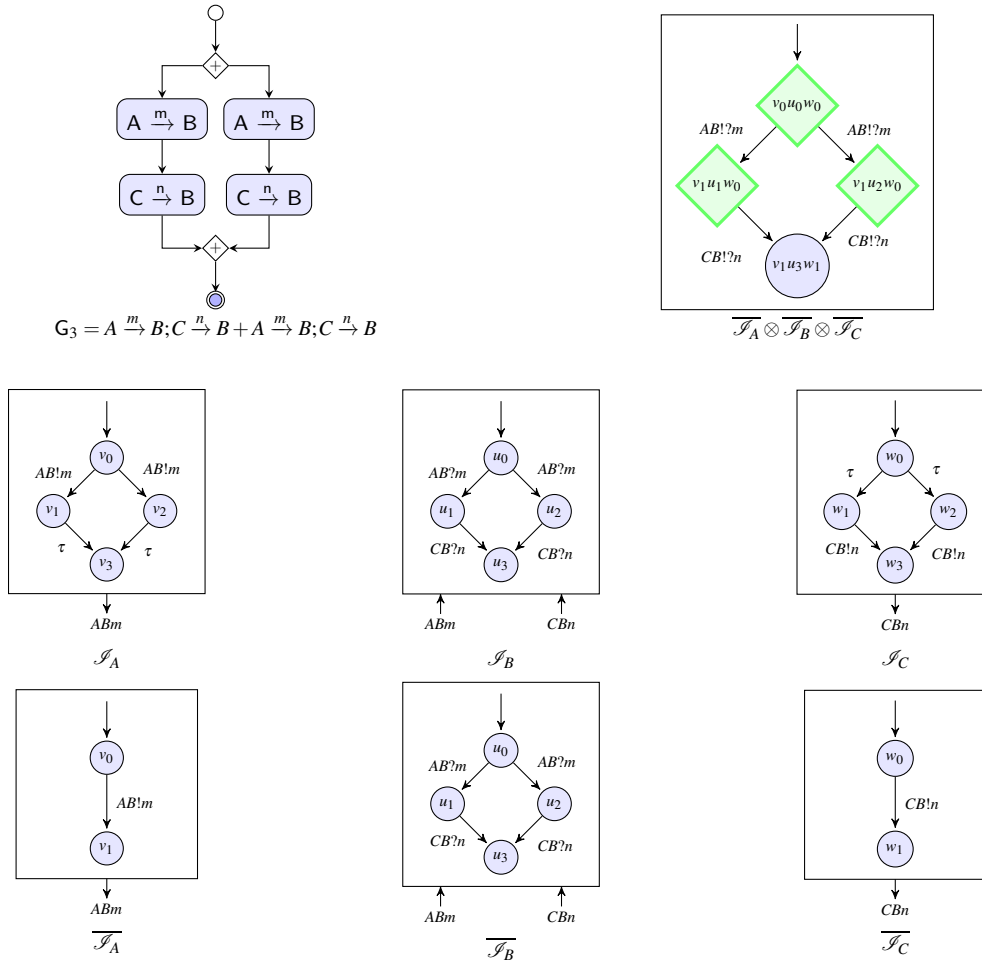
Figure 10: Branching error states in $\otimes_{A\in\mathscr{P}}(G_{3\downarrow A})$

therefore unaware of the choice which is indeed not a "semantic" choice). Here we opted for simplicity; our results can be casted in the more general setting at the cost of increasing the technical complexity.

As a matter of fact, finding removable τ and obtaining refined projections has an exponential theoretical complexity. This is mainly due to the minimisation of finite state machines. We believe that in practice this is not a great problem since in practice minimisation is rather effective. We are developing a prototype tool (https://github.com/haomoons/GIAGG) to conduct experiments and measure how our approach performs in practice available at to support our theory.

References

[1] Jonas Bonér (2018): *Reactive Microsystems - The Evolution Of Microservices At Scale*. O'Reilly. Available at tiny.cc/hk0w5y.

[2] Daniel Brand & Pitro Zafiropulo (1983): *On communicating finite-state machines*. Journal of the ACM (JACM) 30(2), pp. 323–342, doi:10.1145/322374.322380.

[3] Luca De Alfaro & Thomas A Henzinger (2001): *Interface automata*. In: ACM SIGSOFT Software Engineering Notes, 26, ACM, pp. 109–120, doi:10.1145/503209.503226.

[4] Luca De Alfaro & Thomas A Henzinger (2005): *Interface-based design*. In: *Engineering theories of software intensive systems*, Springer, pp. 83–104, doi:10.1016/0022-0000(84)90034-5.

[5] Pierre-Malo Deniélou & Nobuko Yoshida (2012): *Multiparty session types meet communicating automata*. In: *European Symposium on Programming*, Springer, pp. 194–213, doi:10.1007/978-3-642-28869-2_10.

[6] Adrian Francalanza, Claudio Antares Mezzina & Emilio Tuosto (2018): *Reversible Choreographies via Monitoring in Erlang*. In: *Distributed Applications and Interoperable Systems - 18th IFIP WG 6.1 International Conference, DAIS 2018, Held as Part of the 13th International Federated Conference on Distributed Computing Techniques, DisCoTec 2018, Madrid, Spain, June 18-21, 2018, Proceedings*, pp. 75–92, doi:10.1007/978-3-319-93767-0_6.

[7] Roberto Guanciale & Emilio Tuosto (2016): *An Abstract Semantics of the Global View of Choreographies*. In: *Proceedings 9th Interaction and Concurrency Experience, ICE 2016, Heraklion, Greece, 8-9 June 2016.*, pp. 67–82. Available at `https://doi.org/10.4204/EPTCS.223.5`.

[8] Roberto Guanciale & Emilio Tuosto (2017): *Semantics of Global Views of Choreographies*. Journal of Logic and Algebraic Methods in Programming 95. Revised and extended version of [7]. Accepted for publication. To appear; version with proof available at `http://www.cs.le.ac.uk/people/et52/jlamp-with-proofs.pdf`.

[9] Kohei Honda, Nobuko Yoshida & Marco Carbone (2016): *Multiparty Asynchronous Session Types*. Journal of the ACM 63(1), pp. 9:1–9:67, doi:10.1145/2827695. Extended version of a paper presented at POPL08.

[10] Nickolas Kavantzas, David Burdett, Gregory Ritzinger, Tony Fletcher, Yves Lafon & Charlton Barreto (2005): *Web services choreography description language version 1.0*. W3C candidate recommendation 9, pp. 290–313.

[11] Julien Lange, Emilio Tuosto & Nobuko Yoshida (2015): *From Communicating Machines to Graphical Choreographies*. In: *POPL 15*, pp. 221–232, doi:10.1145/2676726.2676964.