ARCHIVIO ISTITUZIONALE
DELLA RICERCA

Alma Mater Studiorum Università di Bologna
Archivio istituzionale della ricerca

Pomsets for Process Management: A Healthcare Case Study

This is the final peer-reviewed author’s accepted manuscript (postprint) of the following publication:

Published Version:
Pal, S., Guanciale, R., Lanese, I., Tuosto, E., Clo, M. (2026). Pomsets for Process Management: A Healthcare Case Study. Springer Science and Business Media Deutschland GmbH [10.1007/978-3-032-11176-0_22].

Availability:
This version is available at: https://hdl.handle.net/11585/1050856 since: 2026-02-26

Published:
DOI: http://doi.org/10.1007/978-3-032-11176-0_22

Terms of use:

Some rights reserved. The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

This item was downloaded from IRIS Università di Bologna (https://cris.unibo.it/).
When citing, please refer to the published version.

(Article begins on next page)

22 April 2026

Pomsets for Process Management: a Healthcare Case Study*

Sourabh Pal[1], Roberto Guanciale[2], Ivan Lanese[1], Emilio Tuosto[3], and Massimo Clo[4]

[1] University of Bologna, Italy; Centre Inria d'Université Côte d'Azur, France
[2] EECS and Digital Futures, KTH Royal Institute of Technology, Sweden
[3] Gran Sasso Science Institute, Italy
[4] Area ICT e Transizione digitale dei servizi al cittadino - Direzione Generale Cura della Persona, Salute e Welfare, Emilia Romagna, Italy

Abstract. Complex coordination protocols are necessary to manage complex organisations. The healthcare management sector is no exception, since different authorities, users, and systems have to interact with each other in order to achieve their organisational goals. In this paper we consider a case study on the authorisation and accreditation of healthcare structures in the Emilia Romagna region in Italy. We specify the case study using *global choreographies* so to enable the analysis of the correctness of its communication patterns using the PomCho tool. This requires to refine PomCho and its underlying theoretical framework. First, we extend PomCho to support not only asynchronous communication, but also synchronous one. Moreover, in both the cases, we provide *a more efficient algorithm* to check closure properties ensuring realisability of choreographies. The new algorithm allows us to check realisability of larger pomsets than before, which makes our approach viable for complex systems such as our case study.

1 Introduction

Distributed systems present a main tension: while the decomposition of functionality across multiple components helps managing complexity and improves efficiency, it also scatters execution contexts and states across the computational environment. This makes it difficult to guarantee global properties. Centralised coordination avoids this problem but introduces performance bottlenecks and single points of failure.

Choreographic specifications [18] offer a high-level way to describe desired interactions by presenting a global view of possible execution traces. This global perspective aims to simplify the modelling of interactions and their ordering constraints. In contrast, the expected behaviour of each component can be captured by formal models such as Communicating Finite State Machines [5] or programming languages featuring message-passing (e.g., Erlang or GoLang).

* Work partially supported by PRIN project FREEDA (CUP: I53D23003550006), by French ANR project SmartCloud ANR-23-CE25-0012, by INdAM – GNCS 2025 project MARQ, code CUP E53C24001950001, by European Union - NextGenerationEU PNRR Mission 4, Component 1, Investment 4.1 (DM 351/2022) - Public Administration, by the PRIN PNRR project DeLICE (F53D23009130001) and by the MUR dipartimento di eccellenza 2023-2027. We thank the reviewers for their comments and suggestions.

Not every choreography admits a correct realisation, namely a realisation as a set of participants whose sequences of possible interactions match the ones specified in the choreography. We provide a language-independent solution based on some realisability conditions on global views. Concretely, we employ partially ordered multisets (pomsets) as an abstract representation of global specifications. A pomset exposes the causal relationships among communication events –namely sends and receives of messages– without prescribing a total order and it is suitable for reasoning about synchronous rendezvous and asynchronous semantics. Our realisability conditions can be checked directly on pomsets, eliminating the need to enumerate interleavings or translate into a particular choreography language.

Contributions and Structure of the Paper. We extend the framework in [14,15] (summarised in Section 2) through the analysis of a comprehensive, realistic case study borrowed from the process management domain (cf. Section 3) whose size challenges both the theory [14] and the implementation in [15]. We refine the pomset framework to encompass synchronous communication, defining new realisability conditions for rendezvous-style send/receive primitives (cf. Section 5). This generalisation ensures that our method applies seamlessly whether components use blocking sends and receives or non-blocking, unordered message queues.

Verifying realisability over large pomsets can be unfeasible. To mitigate this, we introduce an algorithm (cf. Section 6) that identifies a “sufficient prefix”, the largest subset of events that captures all relevant causal constraints, and can prune unnecessary pomsets. In many realistic specifications, these optimisations reduce the verification effort exponentially, making our approach viable for more complex systems (cf. Section 7).

Taken together, these contributions yield a unified, semantically grounded framework for verifying the realisability of scenario-based specifications in both the asynchronous and synchronous settings. By operating directly on pomsets, we enable designers to detect and eliminate coordination mismatches at design time, rather than discovering them as costly implementation bugs.

The tool, all the examples, and the full details on our case study are available in [25].

2 A Bird-Eye View of Choreographies

Choreographies allow one to describe distributed systems composed of multiple participants which interact via message-passing, like formalisms such as *Message Sequence Charts* (MSC) [20,12], multiparty session types [17], and BPMN choreographies [21]. Following the choreographic approach to system design (see, e.g., [18]), distributed systems can be represented using two complementary views, namely a single *global view* or multiple *local views*, one per participant. The global view describes the order in which messages can be exchanged inside a system from a global point of view. It provides both a syntactic and a diagrammatic representation of a distributed communication system. A single global view corresponds to multiple local views, each of which defines the behaviour of a participant. Broadly speaking, the global view is more suitable for specification and understanding, while the local view is closer to the final implementation. Both views focus on communication, abstracting away computation.

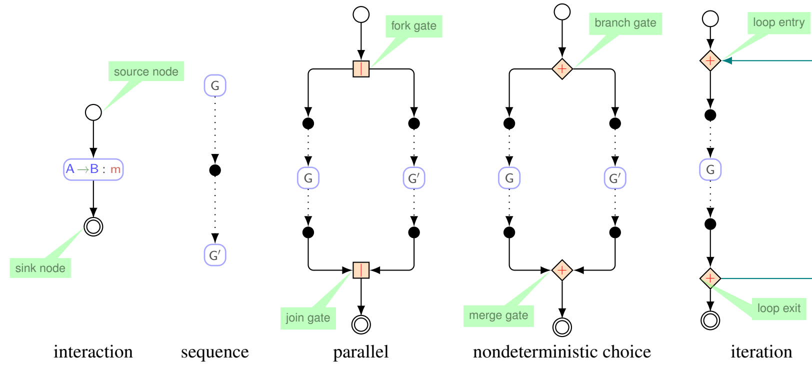


Fig. 1: Diagrammatic Representation of the Syntax of Global Choreographies

Our choreographic framework consists of *global choreographies* (or g-choreographies for short) [30] for global views and *Communicating Finite State Machines* [5] for local views. A g-choreography is a term derivable from the following grammar:

$$\mathsf{G} ::= \mathsf{A} \rightarrow \mathsf{B} : \mathsf{m} \;\big|\; \mathsf{G}; \mathsf{G}' \;\big|\; \mathsf{G} \mid \mathsf{G}' \;\big|\; \mathsf{G} + \mathsf{G}' \;\big|\; \textit{repeat}\ \mathsf{G}$$

The main building block of a g-choreography is an interaction between two participants, written $\mathsf{A} \rightarrow \mathsf{B} : \mathsf{m}$, where participant A sends a message m to participant B, which in turn receives it. G-choreographies can be composed in sequence, in parallel, in nondeterministic choice, or iterated.

Each g-choreography in the syntax above has a graphical representation that can be formally derived by induction on the syntax of the g-choreography. Instead of a formal definition, we show how to achieve this using the sketches in Fig. 1. Intuitively, G can be depicted as a graph rooted in a *source* node $\circ$ and having a single *sink* node $\circledcirc$. Connecting nodes represent interaction, branch or fork points, or else looping points; this is immediate in the representation of interactions in Fig. 1. Besides interaction nodes, our graphical notation uses *gate* nodes ◫ and ◈ to identify fork and branch or iteration as well as their corresponding "closing" points. Gate nodes and dotted edges allow us to compose the diagrams; more precisely, a dotted edge from $\bullet$ to a boxed G (the diagrammatic representation of G) means that the gate connected to the source node of G should be connected to the gate entering in $\bullet$ and that the source node of G is removed; similarly, a dotted edge from a boxed G to $\bullet$ means that the gate arriving in the sink node of G has to be connected to the gate after $\bullet$ and that the sink node of G is removed. For instance, in the graph for the sequential composition, the top-most edge identifies the sink node of G and the other edge identifies the source node of G'.

We will describe the asynchronous and synchronous semantics of choreographies in terms of pomsets in Sections 4 and 5, but the intuition above should be enough to understand the modelling of our case study, described in the next section.

3 Case Study: Process Management in Italian Healthcare

We study and analyse the realisability of a case study on healthcare management, focusing on authorisation and accreditation of the private and public healthcare struc-

tures in the Emilia Romagna region (Italy), which we dub A&A after *A*ccreditation *and A*uthorisation. The protocol describes the interactions among eleven participants, namely the actors and organisations involved in the process. The first task in the protocol is the nomination of the regional coordinator (RCOO) followed by the steps needed to allow RCOO to operate. Afterwards, the protocol describes five concurrent tasks, each of which concerns the iterative execution of a sub-protocol. We focus on the most complex sub-protocol where authorisation or accreditation to the private or public healthcare authorities (LHA) are granted or denied based on investigation reports of the technical teams i.e., evaluators (EV) and expert technicians (ET).

The analysis of A&A based on Choreographic Automata (CAs) [3] and the Corinne tool [22,24] in [23] cannot model concurrent flows of execution in a suitable way. Here, we therefore use *global-choreographies* (g-choreographies) to model A&A and the PomCho tool for the analysis since they natively support concurrency.

The g-choreography modelling A&A takes the form $\mathsf{G}_I; (\mathsf{G}_T \mid \mathsf{G}_A)$ where[5]

$$\begin{aligned}
\mathsf{G}_I = \ & \mathsf{GDH} \to \mathsf{RC} : \mathsf{fndCor}; \mathsf{GDH} \to \mathsf{RCOO} : \mathsf{assRes}; \mathsf{GDH} \to \mathsf{RCOO} : \mathsf{perFunc};\\
& \mathsf{RCOO} \to \mathsf{RC} : \mathsf{mkPropRHS}; \mathsf{RC} \to \mathsf{RHS} : \mathsf{gvCrite}
\end{aligned}$$

$$\begin{aligned}
\mathsf{G}_T = \ & \mathit{repeat}\ \mathsf{RCOO} \to \mathsf{GDH} : \mathsf{proAcc}\\
& \mid \mathit{repeat}\ (\mathsf{RCOO} \to \mathsf{AC} : \mathsf{chkOut}; \mathsf{AC} \to \mathsf{RCOO} : \mathsf{rptinvs})\\
& \mid \mathit{repeat}\ (\mathsf{RCOO} \to \mathsf{GHD} : \mathsf{chkPol}; \mathsf{GHD} \to \mathsf{RCOO} : \mathsf{confChkPol})\\
& \mid \mathit{repeat}\ (\mathsf{RC} \to \mathsf{OTM} : \mathsf{proCriteEv}; \mathsf{RC} \to \mathsf{OTAM} : \mathsf{confCriteEv})
\end{aligned}$$

$$\begin{aligned}
\mathsf{G}_A = \ & \mathit{repeat} \big(\mathsf{LHA} \to \mathsf{GDH} : \mathsf{seekAcc}; \mathsf{GDH} \to \mathsf{RCOO} : \mathsf{forAcc};\\
& \quad \mathsf{RCOO} \to \mathsf{HAS} : \mathsf{perInvs}; \mathsf{HAS} \to \mathsf{RCOO} : \mathsf{rtpinvs}; \big(\\
& \quad\quad \mathsf{RCOO} \to \mathsf{LHA} : \mathsf{decAcc}\\
& \quad\quad +\\
& \quad\quad \big(\mathsf{RCOO} \to \mathsf{LHA} : \mathsf{gnrtAcc} \mid \mathsf{RCOO} \to \mathsf{OTA} : \mathsf{manVer}\big);\\
& \quad\quad \mathsf{G}'_A;\\
& \quad\quad \big(\mathsf{RCOO} \to \mathsf{GDH} : \mathsf{sndPro}\\
& \quad\quad +\\
& \quad\quad \mathsf{RCOO} \to \mathsf{GDH} : \mathsf{rptMon}; \mathsf{GDH} \to \mathsf{LHA} : \mathsf{takAct}\big)\big)\big)
\end{aligned}$$

Fig. 2 reports the g-choreography of A&A[6] using the visual representation[7] of g-choregraphies (cf. Section 2). Both the syntactic and visual representation of A&A show that the most complex part of the protocol is G_A, the g-choreography with highlighted background in Fig. 2. Below we comment on G_A, except for G'_A which is a sequence of interactions immaterial for the rest of the paper.

The g-choreography is a refinement of the CA model in [23] amended from some inaccuracies with the help of domain experts.The CA model was also approximating the actual protocol due to the impossibility of handling the combinatorial explosion

[5] Assume that $_;_$ takes precedence over $_ + _$.

[6] Acronyms are defined according to our English translation of the Italian descriptions in [16,19]; the interested reader can find the correspondence in [23].

[7] See [26] for the g-choreography rendered by PomCho.

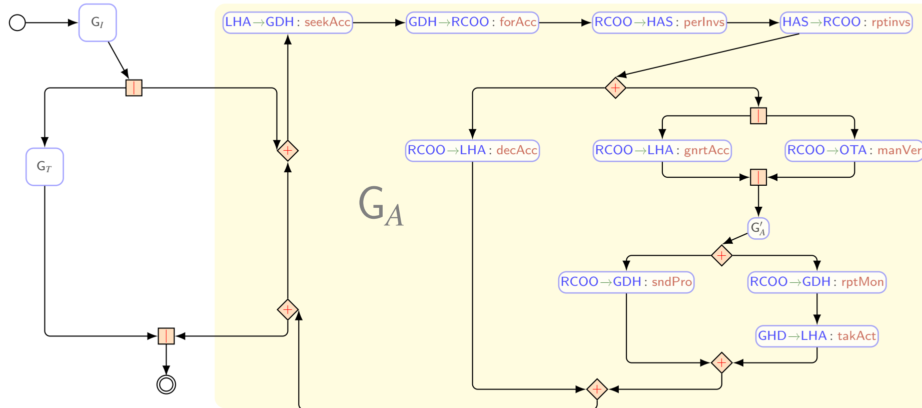


Fig. 2: G-choreography for Regional Coordination for Authorisation and Accreditation

arising from the necessary explicit enumeration of all interleavings of concurrent activities. This problem was avoided in [23] by turning some concurrent computations into non-deterministic choices. Since such computations were loops, it means that the sub-protocols were executed one at the time instead of in interleaving. Besides imprecision, this model yields false positives in the analysis, since non-deterministic composition is subject to stricter conditions than parallel one.

Let us turn our attention to the remarkable structure of G_A.Indeed, G_A is an iterative process where, after the sequence of interactions on top, participants engage in a choice. The right branch is the sequential composition of two activities in parallel, followed by the interactions in G'_A, and finally a nested choice. As we will see in the rest of the paper this is both a source of complexity and the source of subtle glitches in the protocol.

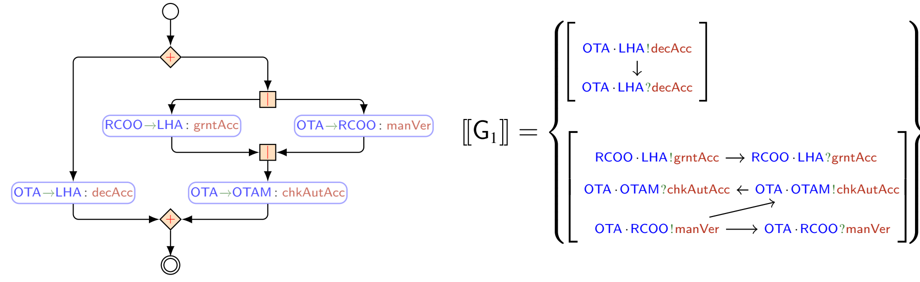
4 Asynchronous Semantics

The asynchronous semantics of a g-choreography is modelled using *partially ordered multisets* (pomsets) [28]. Essentially, a pomset r corresponds to a partial order $\leq_r$, called *happens-before* relation, that specifies the causal dependencies among some events; if $e \leq_r e'$ then e' causally depends e in the pomset r. Instead of the formal definition of pomsets of g-choreographies (which can be found in [30,14]), we give an intuitive presentation of the main concepts.

Our pomsets order events labelled by communication actions. More precisely, actions of the form $\mathsf{A} \cdot \mathsf{B} ! \mathsf{m}$ label events where A sends message m to B, while actions of the form $\mathsf{A} \cdot \mathsf{B} ? \mathsf{m}$ label events where B receives message m from A. The semantics of a g-choreography G is then given by associating the set of pomsets $[\![\mathsf{G}]\!]$ corresponding to the resolutions of each choice in G. This semantics can be defined by induction on the syntactic structure of G. The base case being $\mathsf{A} \to \mathsf{B} : \mathsf{m}$ for which $[\![\mathsf{A} \to \mathsf{B} : \mathsf{m}]\!]$ is a singleton set containing a pomset where an event labelled with $\mathsf{A} \cdot \mathsf{B} ! \mathsf{m}$ (dubbed *output event*) precedes the one labelled $\mathsf{A} \cdot \mathsf{B} ? \mathsf{m}$ (dubbed *input event*). For the inductive cases, the semantics of a choice $\mathsf{G} + \mathsf{G}'$ is the union of the semantics of G and G' (that is $[\![\mathsf{G} + \mathsf{G}']\!] = [\![\mathsf{G}]\!] \cup [\![\mathsf{G}']\!]$); the semantics of $\mathsf{G} ; \mathsf{G}'$ merges the pomsets obtained by the Cartesian product of the pomsets of $[\![\mathsf{G}]\!]$ and $[\![\mathsf{G}']\!]$, adding dependen-

cies that force events of the same participant in G to precede those in G'; the semantics of $\mathsf{G} \mid \mathsf{G}'$ is obtained by merging the pomsets obtained by the Cartesian product of $[\![\mathsf{G}]\!]$ and $[\![\mathsf{G}']\!]$, without adding any dependencies. Finally, the semantics of iteration is given by taking the union of the semantics of the finite unfoldings of G, namely $[\![\mathit{repeat}\ \mathsf{G}]\!] = \bigcup_{n>0} \underbrace{[\![\mathsf{G}]\!];\ldots;[\![\mathsf{G}]\!]}_{n-\text{times}}$.

Example 1. Consider the choreography (inspired by A&A) $\mathsf{G}_1 = \mathsf{G} + \mathsf{G}'$, where $\mathsf{G} = \mathsf{OTA} \to \mathsf{LHA} : \mathsf{decAcc}$ and $\mathsf{G}' = (\mathsf{RCOO} \to \mathsf{LHA} : \mathsf{grntAcc} \mid \mathsf{OTA} \to \mathsf{RCOO} : \mathsf{manVer});$ $\mathsf{OTA} \to \mathsf{OTAM} : \mathsf{chkAutAcc}$. The graphical representation of the g-choreography and its asynchronous semantics are as follows:



The g-choreography G_1 comprises two branches in a choice. The only interaction in the left branch is $\mathsf{OTA} \to \mathsf{LHA} : \mathsf{decAcc}$, while the right branch consists of the parallel interactions $\mathsf{RCOO} \to \mathsf{LHA} : \mathsf{grntAcc}$ and $\mathsf{OTA} \to \mathsf{RCOO} : \mathsf{manVer}$, followed by the interaction $\mathsf{OTA} \to \mathsf{OTAM} : \mathsf{chkAutAcc}$.

Hence, $[\![\mathsf{G}_1]\!]$ consists of a pomset where LHA must consume the decAcc from OTA and another where participants LHA and RCOO may receive messages grntAcc and manVer in any order. Note that the sending of message chkAutAcc depends only on the sending of message manVer, since they are the only interactions in different components of the sequential composition done by the same participant. ◇

A *linearisation* of a pomset r is a sequence $\{e_i\}_i$ of its events that respects the happens-before relation of r, namely if $i < j$ then $e_j \not\leq_r e_i$. The language of a g-choreography is the union over of the sequences of labels of all the *linearisations* of the pomsets of its semantics.

A g-choreography among participants $\mathsf{A}_1, \ldots, \mathsf{A}_n$ is implemented via a *communicating system* [5], that is an assignment to each participant A_i of a *communicating finite-state machine* (CFSM) [5] (i.e., a finite-state automaton whose transition labels are communication actions of A_i). In our asynchronous model of communicating systems, a configuration is a tuple $(q_1, \ldots, q_n, M)$, where q_i is the current state of the CFSM of A_i and M is an unordered multiset of (pending) messages. The firing of a transition $q_i \xrightarrow{\mathsf{A}_i \cdot \mathsf{B}!\mathsf{m}} q_i'$ yields the new configuration $(q_1, \ldots, q_i', \ldots, q_n, M \uplus \{\mathsf{A}_i \cdot \mathsf{B}!\mathsf{m}\})$, where $\uplus$ denotes multiset union. If $M = M' \uplus \{\mathsf{B} \cdot \mathsf{A}_i!\mathsf{m}\}$ then the firing of a transition $q_i \xrightarrow{\mathsf{B} \cdot \mathsf{A}_i?\mathsf{m}} q_i'$ yields the new configuration $(q_1, \ldots, q_i', \ldots, q_n, M')$. The language of a system of CFSMs is the set of all sequences of labels on runs starting from its initial state.

Some g-choreographies are not properly implementable. For example, in G_1 (from Example 1), participants must agree on whether to take the left or the right branch. However, RCOO is not involved in the left branch, and can decide autonomously to move on the right branch sending message grntAcc, while participant OTA may send decAcc taking the left branch. This leads to an erroneous computation, not expected from the choreography G_1. Formally, a g-choreography G is *realisable* if there exists a communicating system S such that $\mathcal{L}(S) = \mathcal{L}(\mathsf{G})$, where $\mathcal{L}(\mathsf{G})$ is the set of all linearisations of the pomsets in $[\![\mathsf{G}]\!]$. This ensures that every run of S conforms to at least one linearisation of a pomset of the g-choreography and vice-versa.

Two sufficient closure conditions for the realisability of a set of pomsets under the asynchronous semantics have been described in [14]. We introduce some auxiliary notation to state them (see Example 2 for concrete uses of the notations below). A pomset r_1 is a prefix of a pomset r_2 if its events are a subset of the events of r_2 and it is downward closed w.r.t. the happens-before relation of r_2 (i.e., if $e \leq_{r_2} e'$ and $e' \in r_1$ then $e \in r_1$) and equipped with the projection of the happens-before relation of r_1. Under the asynchronous semantics, a pomset is *well-formed* if[8]:

- each output event $\mathsf{A} \cdot \mathsf{B} ! \mathsf{m}$ (resp. input event $\mathsf{A} \cdot \mathsf{B} ? \mathsf{m}$) has at most (resp. exactly) one immediate successor input event $\mathsf{A} \cdot \mathsf{B} ? \mathsf{m}$ (resp. immediate predecessor output event $\mathsf{A} \cdot \mathsf{B} ! \mathsf{m}$);
- whenever an event e is an immediate predecessor of another event e', then e and e' are labelled either by communication actions of the same participant or by matching output and input actions; and
- for any two distinct outputs e and e' with the same label and e' causally depending on e, no immediate-successor matching input of e' may precede any immediate-successor matching input of e.

The projection of a pomset r onto a participant A, denoted $r\lfloor_{\mathsf{A}}$, is the sub-pomset consisting of all events in r that are labeled with communication actions involving A. Given a set of pomsets R over participants $\mathcal{P}$, we define the set of *branch assignments on* R as the set Π_R of functions such that $\pi(\mathsf{A}) \in \{r\lfloor_{\mathsf{A}} \mid r \in R\}$ for $\mathsf{A} \in \mathcal{P}$. Intuitively, a branch assignment $\pi \in \Pi_R$ chooses, for each to participant $\mathsf{A} \in \mathcal{P}$, a pomset $r \in R$ (corresponding to a resolution of a g-choreography) and assigns $r\lfloor_{\mathsf{A}}$ to A. Note that π can choose different resolutions for different participants. We use $\cup_{\mathsf{A}} \pi(\mathsf{A})$ for the pomset obtained by merging the pomsets in the codomain of π. Finally, the *inter-participant closure* of a pomset, denoted $\Box(r)$, is the set of all pomsets that can be constructed by adding dependencies to connect individual output events with their corresponding input events across participants. The resulting pomsets in $\Box(r)$ represent all valid ways of linking outputs to inputs. A pomset r is *more permissive than* another pomset r' if r has the same events and labels of r' and its happens-before relation is a subset of the one of r' (namely, r imposes fewer causal dependencies than r'). The closure conditions CC2 and CC3 below jointly guarantee relisability. Intuitively, CC2 requires that if a set of executions (i.e., a pomset) cannot be taken apart by any participant from the executions (i.e., pomsets) of R, then those executions must be part of R.Intuitively, CC3 requires that if a set of executions (i.e., a pomset) can be consistently represented as a prefix R'

[8] These conditions are adapted from the ones in [1].

of another pomset in R, indicating that some participants will terminate earlier while matching the executions of R, then those prefixes R' must be part of R.

Definition 1 (Closure conditions). *A set of pomsets R satisfies* CC2 *if for all well-formed $r \in \{\Box(\cup_{\mathsf{A}} \pi(\mathsf{A})) \mid \pi \in \Pi_R\}$ there is $r' \in R$ that is at least as permissive as r; R satisfies* CC3 *if for all R' set of prefixes of R and all well-formed $r \in \{\Box(\cup_{\mathsf{A}} \pi(\mathsf{A})) \mid \pi \in \Pi_{R'}\}$, there is a prefix r' of a pomset in R such that r' is at least as permissive as r.*

The closure conditions in Definition 1 are adapted from the realisability criteria for MSCs [1] and avoid the explicit computation of the languages of the choreography.

Example 2. We illustrate our closure conditions on the semantics of G_1 from Example 1. Since $[\![\mathsf{G}_1]\!]$ consists of two pomsets involving participants RCOO, LHA, OTA, and OTAM}, Π_R contains 2^4 functions, one for each possible combination of the projections of these pomsets on the four participants.

It is straightforward to see that only two functions of Π_R have a well-formed inter-participant closure: the ones where all participants have chosen the same pomset of R. For example, in one of the other cases, if the participant RCOO chooses the first pomset and OTA chooses the second one, the input $\mathsf{OTA} \cdot \mathsf{RCOO?manVer}$ has no corresponding output. Since the two well-formed pomsets coincide with the choreography semantics, the CC2 condition is met.

However, CC3 is not satisfied. In fact, let $\pi(\mathsf{RCOO}) = [\mathsf{RCOO} \cdot \mathsf{LHA!grntAcc}]$ (i.e., prefix of the first branch), $\pi(\mathsf{LHA}) = [\mathsf{RCOO} \cdot \mathsf{LHA?grntAcc}]$ (i.e., first branch), $\pi(\mathsf{OTA}) = [\mathsf{OTA} \cdot \mathsf{LHA!decAcc}]$ and $\pi(\mathsf{OTAM}) = []$ (i.e., both second branch). The inter-participant closure yields the well-formed pomset $r = [\mathsf{RCOO} \cdot \mathsf{LHA!grntAcc} \to \mathsf{RCOO} \cdot \mathsf{LHA?grntAcc} \quad \mathsf{OTA} \cdot \mathsf{LHA!decAcc}]$, however no prefix of pomsets in R is more permissive than r. Pomset r (as any other where RCOO chooses one branch and OTA chooses the other one) demonstrate the lack of direct or indirect (via other participants) communication between OTA and RCOO to select the proper branch. ◇

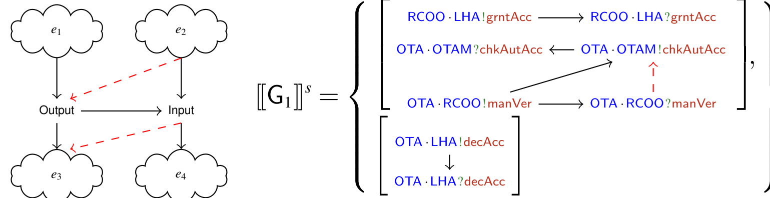
Note that condition CC3 does not imply condition CC2, as proved by the counterexample in [27].

5 Synchronous Semantics

In order to cope with different communication models, we extend the framework to support synchronous message passing. The synchronous semantics of a g-choreography G, denoted as $[\![\mathsf{G}]\!]^s$, is defined by induction on the syntactic structure of G. The only difference w.r.t. the asynchronous semantics is the case for $\mathsf{G};\mathsf{G}'$, which adds the dependencies necessary to model the handshake between sender and receiver: (1) events of a participant in G must precede the events of the same participant in G'; (2) events in G' that depend on an output in G must also depend on the corresponding input in G; (3) events in G preceding an input in G' must also precede the corresponding output in G'.

The synchronous semantics of a choreography can be computed by adding the missing dependencies to its asynchronous semantics: for every pomset and for every output e and its matching successor input e' every predecessor of e' must be a predecessor of e and every successor of e must be successor of e'. Example 3 below shows a pictorial representation of the translation from asynchronous to synchronous semantics.

Example 3. The synchronous semantics of the g-choreography of Example 1 is $[\![\mathsf{G}_1]\!]^s$, which is as follows:



In the synchronous setting, every successor of the event OTA · RCOO!manVer (here just OTA · OTAM!chkAutAcc) must be a successor of OTA · RCOO?manVer as well, hence the additional dashed red arrow in picture. The events corresponding to the interaction OTA→OTAM : chkAutAcc remain however independent from those events for RCOO→LHA : grntAcc since no participant is involved in both the interactions. Moreover, the events for RCOO→LHA : grntAcc and the ones for OTA→RCOO : manVer are independent of each others since they belong to different threads. ◇

The choice of modeling interactions also in the synchronous case with two distinct events (send and receive) while preserving independence of concurrent threads allows us to reuse the same theory and machinery of the asynchronous semantics, which is grounded on the work in [1] on Message Sequence Charts. However this results in each interaction (e.g., OTA→RCOO : manVer) being modeled by a two-way handshake (e.g., OTA · RCOO!manVer and OTA · RCOO?manVer) that cannot be interleaved by other messages of the same participants on the same thread (e.g., OTA · OTAM!chkAutAcc), but can be interleaved by other messages on other threads (e.g. RCOO · LHA!grntAcc). An alternative semantics could maintain distinct events per interaction, thus still benefiting from the asynchronous machinery, while also preventing interleaving of other threads between the send and receive events of a single interaction. This would require generating two pomsets per interaction, each enriched with specific dependencies to reflect thread-level ordering. E.g., we would have: one pomset with the additional edge OTA · RCOO?manVer → RCOO · LHA!grntAcc and one pomset with the additional edge RCOO · LHA?grntAcc → RCOO · LHA!manVer.

To avoid introducing new labels, the two-way handshake scheme described above is reflected in the LTS for the synchronous semantics of communicating systems by "overloading" the communication labels to let them represent handshaking. More precisely, the realisation of the synchronous semantics of a g-choreography involving participants $\mathsf{A}_1, \ldots, \mathsf{A}_n$ can be attained through a *synchronous communicating system*, where a configuration is a tuple of length n whose i-th element is either the current state of (the CFSM of) A_i or the *handshake point* $[q_i l]$ with q_i a state of A_i and l a label on a transition from q_i; call *stable* configurations that do not have handshake points. Let $(q_1, \ldots, q_n)$ be a stable configuration such that $q_i \xrightarrow{\mathsf{A_i \cdot A_j ! m}} q'_i$ is a transition of A_i and $q_j \xrightarrow{\mathsf{A_i \cdot A_j ? m}} q'_j$ is a transition of A_j then the synchronous LTS will have the transitions

$$\begin{aligned}(q_1, \ldots, q_i, \ldots, q_j, \ldots, q_n) &\xrightarrow{\mathsf{A_i \cdot A_j ! m}} (q_1, \ldots, [q_i \mathsf{A_i \cdot A_j ! m}], \ldots, [q_j \mathsf{A_i \cdot A_j ? m}], \ldots, q_n) \\ &\xrightarrow{\mathsf{A_i \cdot A_j ? m}} (q_1, \ldots, q'_i, \ldots, q'_j, \ldots, q_n)\end{aligned}$$

(and similarly for the configuration $(q_1,\dots,q_j,\dots,q_i,\dots,q_n)$). The language of a system of CFSMs is the set of all sequences of labels corresponding to the transitions of runs from the initial configuration.

In order to check realisability in the synchronous setting, we define synchronous well-formedness of pomsets: (i) the pomset is well-formed w.r.t. the asynchronous semantics; (ii) each output event $\mathsf{A}\cdot\mathsf{B}!\mathsf{m}$ has exactly one matching immediate-successor input $\mathsf{A}\cdot\mathsf{B}?\mathsf{m}$. Since the verification conditions over pomsets are parametrised on well-formedness, to check realisability in the synchronous case it is enough to check the closure conditions (cf. Definition 1) using the synchronous well-formedness.

Notice that some choreographies are realisable in the synchronous model while they are not realisable in the asynchronous one. For instance the g-choreography of Example 1 does not satisfy CC3 in the asynchronous case as seen in Example 2, but it does in the synchronous semantics. In fact, the counterexample used for the asynchronous case $r = [\mathsf{RCOO}\cdot\mathsf{LHA}!\mathsf{grntAcc} \to \mathsf{RCOO}\cdot\mathsf{LHA}?\mathsf{grntAcc} \quad \mathsf{OTA}\cdot\mathsf{LHA}!\mathsf{decAcc}]$ is not well-formed in the synchronous model, since the output $\mathsf{OTA}\cdot\mathsf{LHA}!\mathsf{decAcc}$ has no corresponding input, and therefore it is not considered for the closure condition in the synchronous case. Intuitively, the g-choreography of Example 1 can be implemented in the synchronous model since OTA and RCOO must synchronise in the right thread of the right branch, making impossible for them to choose two different branches.

Some choreographies are also not realisable in the synchronous model. For instance, replacing the interaction $\mathsf{OTA}{\to}\mathsf{LHA}\colon \mathsf{decAcc}$ with $\mathsf{OTAM}{\to}\mathsf{LHA}\colon \mathsf{decAcc}$ in the g-choreography in Example 1 yields the g-choreography G'_5 with semantics

$$[\![\mathsf{G}'_5]\!]^s = \left\{ \left[\begin{array}{ccc} \mathsf{RCOO}\cdot\mathsf{LHA}!\mathsf{grntAcc} & \longrightarrow & \mathsf{RCOO}\cdot\mathsf{LHA}?\mathsf{grntAcc} \\ \mathsf{OTA}\cdot\mathsf{OTAM}?\mathsf{chkAutAcc} & \longleftarrow & \mathsf{OTA}\cdot\mathsf{OTAM}!\mathsf{chkAutAcc} \\ & & \uparrow \\ \mathsf{OTA}\cdot\mathsf{RCOO}!\mathsf{manVer} & \longrightarrow & \mathsf{OTA}\cdot\mathsf{RCOO}?\mathsf{manVer} \end{array} \right] , \quad \left[\begin{array}{c} \mathsf{OTAM}\cdot\mathsf{LHA}!\mathsf{decAcc} \\ \downarrow \\ \mathsf{OTAM}\cdot\mathsf{LHA}?\mathsf{decAcc} \end{array} \right] \right\}$$

where the left pomset is as in $[\![\mathsf{G}_1]\!]^s$, whereas the other consists of $\mathsf{OTAM}\cdot\mathsf{LHA}!\mathsf{decAcc}$ followed by $\mathsf{OTAM}\cdot\mathsf{LHA}?\mathsf{decAcc}$. Since there is no coordination among the participants OTAM and LHA in the latter pomset with the participants OTA and RCOO of the first interaction of a thread of the former pomset, the execution can result in OTAM and LHA taking a branch different than the one taken by OTA and RCOO. Consider the prefixes $\pi(\mathsf{RCOO}) = [\mathsf{OTA}\cdot\mathsf{RCOO}?\mathsf{manVer}]$ and $\pi(\mathsf{OTA}) = [\mathsf{OTA}\cdot\mathsf{RCOO}!\mathsf{manVer}]$ of the projections of the first branch and the projections of the second branch $\pi(\mathsf{LHA}) = [\mathsf{OTAM}\cdot\mathsf{LHA}?\mathsf{decAcc}]$ and $\pi(\mathsf{OTAM}) = [\mathsf{OTAM}\cdot\mathsf{LHA}!\mathsf{decAcc}]$: there is no prefix of R that is at least as permissive as the well-formed pomset in the inter-participant closure of $\bigcup_{\mathsf{A}\in\mathcal{P}} \pi(\mathsf{A})$, namely

$$\begin{array}{c} [\mathsf{OTA}\cdot\mathsf{RCOO}!\mathsf{manVer} \to \mathsf{OTA}\cdot\mathsf{RCOO}?\mathsf{manVer} \\ \mathsf{OTAM}\cdot\mathsf{LHA}!\mathsf{decAcc} \to \mathsf{OTAM}\cdot\mathsf{LHA}?\mathsf{decAcc}] \end{array}$$

6 Verifying Realisability

The closure conditions must be verified on finite pomsets. Therefore, we consider a bounded unrolling of loops in the implementation, limited to two iterations. This ap-

Algorithm 1.1: A new algorithm for CC2

```
1  CC2(R, 𝒫):
2      let R_A = {r↾_A | r ∈ R} for A ∈ 𝒫
3      choose A ∈ 𝒫
4      (R′, 𝒫′) = (R_A, {A})
5      while 𝒫 \ 𝒫′ ≠ ∅
6          choose A ∈ 𝒫 \ 𝒫′
7          R″ = ∅
8          for r′ ∈ R′, r ∈ R_A :
9              for r″ ∈ □(r ∪ r′):
10                 if ∀A′·A″?m ∈ r″ such that (A′, A″) ∈ (𝒫′ × {A}) ∪ ({A} × 𝒫′)
11                    ∃ immediate predecessor A′·A″!m ∈ r″ :
12                    # For synchronous case also check
13                    # ∀A′·A″!m ∈ r″ ∃ immediate successor A′·A″?m ∈ r″
14                      R″ = R″ ∪ {r″}
15         (R′, 𝒫′) = (R″, 𝒫′ ∪ {A})
16     return ∀r ∈ R′ ∃r′ ∈ R as permissive as r
```

proach allows us to detect coordination issues such as: a message from one loop iteration being confused with a message from the next; participants disagreeing on the number of iterations executed; a message occurring after the loop being mistaken for one within the loop; and events from a loop iteration being reordered with respect to messages in the subsequent iteration. The main source of complexity in verifying closure conditions is that we have to check prefix-language inclusion and in the size of inter-participant closures. The next two sections tackle these problems.

6.1 Avoiding pomset compositions explosion

Given a set of pomsets R over participants $\mathcal{P}$, the main challenge in checking CC2 arises from the exponential growth of the number of elements in the set of branch assignments Π_R with the size of $\mathcal{P}$. For example, consider the choreography $G_{xy} = G_x + G_y$, where:

$$G_x = A_1 \to A_2 : x; A_2 \to A_3 : x \ldots ; A_{n-1} \to A_n : x$$
$$G_y = A_1 \to A_2 : y; A_2 \to A_3 : y; \ldots ; A_{n-1} \to A_n : y$$

Here, n participants sequentially exchange either x or y. There are two possible projections per participant, one for each branch, so the set Π_R consists of 2^n combinations.

The key observation is that most of the pomsets in Π_R are either non-well-formed or share the same largest well-formed prefix relevant for CC3. Moreover, suppose we are given two pomsets r_A and r_B from the projections of participants $A, B \in \mathcal{P}$. If the dependencies in the union of r_A and r_B cannot be augmented to match an input of B from A, then the well-formedness condition will be violated in every $\pi \in \Pi_R$ containing r_A and r_B, regardless of the pomsets chosen for the other participants.

These observations lead to the new procedure for CC2 in Algorithm 1.1, which checks well-formedness incrementally, pruning invalid partial candidates instead of computing the full Π_R explicitly. Line 4 initialises two variables: variable $\mathcal{P}'$ keeps

track of the set of already processed participants, while R' keeps track of the valid partial candidates: pomsets in $\{\Box(\cup_{\mathsf{A}}\pi(\mathsf{A})) \mid \pi \in \Pi_{R\downharpoonright_{\mathcal{P}'}}\}$ that are well-formed w.r.t. $\mathcal{P}'$.

In practice, the algorithm starts with an initial participant (line 3) and selects the next participant (line 6) as the one that can resolve the largest number of pending input/output dependencies in R'. This heuristic is also applied in the CC3 check, where instead of pruning non-well formed pomsets we keep only the largest well-formed prefixes w.r.t. the processed participants.

When processing the next participant A, the algorithm merges every partial candidate with every projection of A (line 8), iterates over their interparticipant closure (line 9), and adds only the resulting new candidates that satisfy well-formedness w.r.t. all events that involve A (checked on lines 10-11 through a condition that implies well-formedness due the fact that pomsets are obtained by the inter-participant closure of the projection of the well-formed pomsets of R). Finally, the algorithm verifies the existence of a more permissive pomset in R for every final candidate in R'.

The heuristic above is relevant since the order in which participants are chosen (line 6) significantly affects the efficiency of pruning. For instance, if in G_{xy} above participants are chosen in order $\mathsf{A}_1, \mathsf{A}_2, \ldots, \mathsf{A}_n$ then the size of R' remains 2 throughout the execution. In contrast, a processing order like $\mathsf{A}_1, \mathsf{A}_3, \ldots, \mathsf{A}_{2m+1}, \ldots$ can cause the size of R' to grow up to 2^m pomsets after m iterations, due to the lack of direct interactions between the first m selected participants.

6.2 Avoiding prefix explosion

There is a combinatorial blowup to check CC3 due to the examination of all the prefixes of the pomsets in the choreography's semantics, whose number grows exponentially with the number of events across concurrent threads. For example, given the partial order $[e_1 \to e_2 \quad e'_1 \to e'_2]$, there are seven distinct non-empty proper prefixes: $[e_1]$, $[e'_1]$, $[e_1 \quad e'_1]$, $[e_1 \to e_2]$, $[e'_1 \to e'_2]$, $[e_1 \to e_2 \quad e'_1]$, and $[e_1 \quad e'_1 \to e'_2]$. This becomes especially costly in asynchronous semantics, since the sequential composition of interactions targeting different receivers makes the corresponding receive events concurrent.

The main observation is that if r is a counterexample (i.e., r is well-formed and no prefix of R is more permissive than r), then every well-formed $r' \in \{\Box(\cup_{\mathsf{A}}\pi(\mathsf{A})) \mid \pi \in \Pi_{R'}\}$ such that r is a prefix of r' is also a counterexample. Moreover, every pomset admits a unique largest well-formed prefix, which can be obtained by removing all non-well-formed events together with their descendants. Based on this insight, the new approach in Algorithm 1.2 avoids explicitly enumerating all prefixes of R. Instead, it performs the interparticipant closure $R'' = \{\Box(\cup_{\mathsf{A}}\pi(\mathsf{A})) \mid \pi \in \Pi_R\}$, then it computes the largest well-formed prefixes R''' of R'', and finally checks that, for every $r''' \in R'''$, there exists a prefix of R that is more permissive than r'''.

In the actual algorithm, similarly to CC2, we do not explictely compute the interparticipant closure or the largest well-formed prefix. Instead, we build the largest prefixes incrementally. The variable R' keeps track of the largest prefixes of $\{\Box(\cup_{\mathsf{A}}\pi(\mathsf{A})) \mid \pi \in \Pi_{R\downharpoonright_{\mathcal{P}'}}\}$ that are well-formed w.r.t. $\mathcal{P}'$. When processing a next participant A, we merge R' with the projections of A and we extend the candidates with the largest well-formed prefixes. Computing the largest well-formed prefix of a pomset (lines 10-14) is straightfor-

Algorithm 1.2: A new algorithm for CC3

```
1  CC3(R, 𝒫):
2      let R_A = {r↾_A | r ∈ R} for A ∈ 𝒫
3      choose A ∈ 𝒫
4      (R′, 𝒫′) = (R_A, {A})
5      while 𝒫 \ 𝒫′ ≠ ∅
6          choose A ∈ 𝒫 \ 𝒫′
7          R″ = ∅
8          for r′ ∈ R′, r ∈ R_A :
9              for r″ ∈ ☐(r ∪ r′):
10                 while ∃e = A′·A″?m ∈ r″ s.t. (A′,A″) ∈ (𝒫′ × {A}) ∪ ({A} × 𝒫′)
11                         and ∄ immediate predecessor A′·A″!m ∈ r″:
12                         # For synchronous case also check
13                         # e = A′·A″!m ∈ r″ and ∄ imm. succ. A′·A″?m ∈ r″:
14                         r″ = r″ \ subgraph from e
15                 R″ = R″ ∪ {r″}
16         (R′, 𝒫′) = (R″, 𝒫′ ∪ {A})
17     return ∀r ∈ R′ ∃r′ ∈ R as permissive as r
```

ward: we iteratively remove non-well-formed events that involve A along with all events causally dependent on them (i.e., input events without a corresponding immediate predecessor output, and in the synchronous case, output events without a corresponding immediate successor input), until only well-formed events remain.

7 Evaluation

We integrated Algorithms 1.1 and 1.2 in a new version of PomCho dubbed PomCho+. We now describe the application of PomCho+ to A&A and compare it to PomCho.

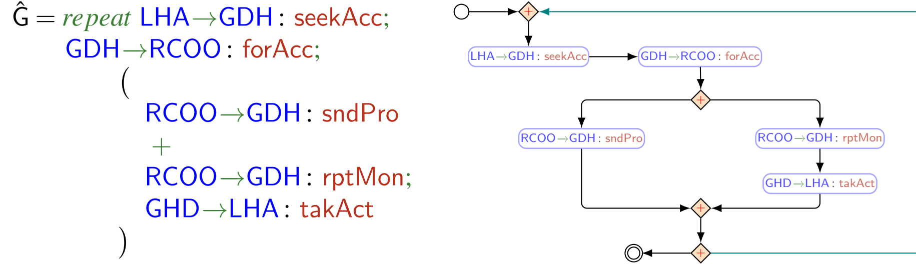
7.1 Verifying the closure conditions on A&A

The application of our analysis to A&A shows that it is realisable neither under the asynchronous nor the synchronous semantics. The main reason for this is the choice at the end of the loop in G_A (cf. Figure 2, or the extract A&A below). One branch of the choice contains RCOO→GDH : sndPro, while the other one contains the interactions RCOO→GDH : rptMon; GDH→LHA : takAct. Note that participant LHA is only involved in the latter case, hence if the other branch is taken LHA does not know whether the loop has ended or not. This issue is made worst since LHA is the sender in the first transition of the loop, hence it does not know whether to trigger a new iteration by sending a seekAcc message to GDH or to wait for a message in the current iteration.

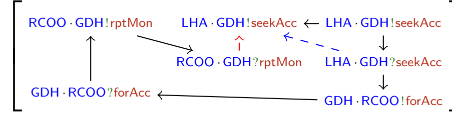
The above is the main issue flagged by our analysis, but the analysis in [23] hid it among many false positives given by the treatment of parallel composition and late join (cf. [23, Sections 5-6]). Domain experts confirmed that the CA model in [23] was not faithful to the specification which indeed requires a notification to LHA on both the branches. Adding such notification makes the protocol realisable under both the synchronous and the asynchronous semantics. The realisable protocol is available in [25].

The error was induced by the fact that in case of a report (right branch) the LHA needs to react. Hence this communication is more emphasised in the informal description [19] on which our model is based. Even if the analysis has not revealed a real bug in the protocol, it has highlighted an error in the model contributing to its refinement. The absence of false positives was also key to highlight real issues.

We give a snippet $\hat{\mathsf{G}}$ of our model that constitutes a minimal example illustrating the problem described above. This allows us to identify a reasonably small pomset flagging the issue instead of analysing the large one corresponding to our full case study.



The counterexample of the semantics $[\![\hat{\mathsf{G}}]\!]$ and $[\![\hat{\mathsf{G}}]\!]^s$ (blue edge is only part of synchronous semantics) of the g-choreography $\hat{\mathsf{G}}$ is



The pomset above is constructed from the projection of LHA that corresponds to the left branch for the first loop iteration, while all projections of the other participants correspond to the right branch. This counterexample shows that LHA can initiate the second iteration (i.e., second LHA · GDH!seekAcc) without waiting the message takAct from GDH. Therefore, the temporal dependency between RCOO · GDH?rptMon and the second LHA · GDH!seekAcc (i.e., the missing red edge) is violated. For this reason, there is no pomset in $[\![\hat{\mathsf{G}}]\!]$ or $[\![\hat{\mathsf{G}}]\!]^s$ that has a prefix more permissive than this counterexample.

7.2 Benchmarking our algorithms

We benchmark our algorithms against those in [13] on examples from previous sections, including A&A. Previous work only supported asynchronous semantics, so we have no synchronous reference. Table 1 reports the results of our benchmarks, conducted on a Macbook Air M2, operating system Ubuntu 22.04.5 LTS with 64 bit, Kernel: Linux 5.19.5-10-asahi, arm64 architecture, and 8GB RAM. Each row corresponds to the g-choreography in the first column. The remaining columns report the results on the algorithms considered (sub-columns Old and New for the asynchronous algorithm in [30] and ours, respectively, and Sync for our synchronous algorithm). More precisely, # WF yields the number of analysed combinations of well-formed pomsets; # Pomsets the number of pomsets in the inter-participant closure for CC2 and of prefixes analysed for

	# WF			# Pomsets			# CEx			Time (ms)		
	Old	New	Sync	Old	New	Sync	Old	New	Sync	Old	New	Sync
G_1	16	2	2	2 26	2 3	2 2	0 3	0 1	0 0	8 43	6 10	7 9
$\hat{G}$	216	12	12	7 56	7 7	7 7	1 20	1 3	0 1	228 1091	91 133	130 145
G_{xy}	32K	2	2	2 ⋆	2 2	2 2	0 ⋆	0 0	0 0	35K ⋆	139 809	157 1165
G_5	1	1	1	1 441	1 1	1 1	0 0	0 0	0 0	26 22K	24 14	28 29
A&A	⋆	480	480	⋆ ⋆	224 112	224 112	⋆ ⋆	32 80	16 8	⋆ ⋆	863K 439K	968K 445K
A&A †	⋆	416	416	⋆ ⋆	192 80	192 64	⋆ ⋆	0 0	0 0	⋆ ⋆	1055K 402K	1048K 334K

Table 1: Benchmarks (⋆ indicates timeout without success after 1800 seconds)

CC3 (highlighted); # CEx the number of pomsets that do not satisfy the condition for CC2 and CC3 while Time yields the total running time to check CC2 and CC3.

For small examples, such as G_1 of Example 1 and $\hat{G}$ of Section 7.1, the new algorithms achieve speedups ranging from $1.4\times$ to $8.4\times$, due to the reduced number of prefixes analysed and the preventive pruning of non-well formed candidates. Notice that G_1 is realisable in the synchronous model only.

We assessed the contribution of the individual optimisations via dedicated examples. The g-choreography G_{xy} of Section 6.1 with $n = 15$ allows us to evaluate the impact of the number of participants: PomCho+ achieves a speedup of $272\times$ for CC2 since it maintains only two well-formed pomsets during the whole execution, while the size of Π_R computed with PomCho is $2^{15} = 32768$. It is not possible to compute the speedup for CC3, since the size of the set of branch assignments makes PomCho timeout on CC3. We have experimented with different values of n. The old algorithm can compute neither CC2 nor CC3 when $n > 10$. The proposed algorithm checks both CC2 and CC3 for all $n \leq 35$ and times-out for $n = 36$ with a timeout of 1800. The g-choreography G_5 is a variant of G_{xy} from Section 6.1, where in G_x (resp. G_y) participant A sequentially sends messages x_1, x_2, x_3, x_4, and x_5 (resp. y_1, y_2, y_3, y_4, and y_5) to B. As described in Section 6.2, the semantics of this choreography consists of one pomset with 20 distinct events with 10 events in each thread. This pomset has 441 prefixes, which have to be computed explicitly by the old approach. The new algorithm is able to maintain only one single largest prefix and achieve a speedup of $1603\times$.

The old implementation is incapable to analyse A&A. Since there are on average about 16 branches for each of the 11 participants, the size of the set of branch assignments would be $16^{11} \approx 1.7 \times 10^{13}$. Each pomset would also have a large number of prefixes, due to the four threads. In contrast, PomCho+ maintains up to 480 partial well-formed pomsets while analysing CC2, and 112 prefixes while analysing CC3. Similar considerations arise for A&A †, which is A&A updated so to make it realisable.

8 Conclusion, Related, and Future Work

This work builds on [23], where the analysis on the case study in Section 3 uses CAs [3] and the Corinne tool [22,24]. A main drawback of CAs (and, consequently, of Corinne) is that concurrency is modelled by explicitly representing all interleavings. This makes both the diagrammatic representation and the analysis quickly unfeasible for highly concurrent cases, such as A&A. The problem was avoided in [23] by turning some concurrent computations into non-deterministic choices. This approach has two drawbacks;

firstly it does not faithfully model A&A, secondly, it produces false positives since non-deterministic choice is subject to stricter conditions than parallel composition.

Hence, to provide a satisfactory analysis of A&A, we turned to g-choreographies that compactly capture parallelism. This is convenient also because computations of g-choreographies can be compactly expressed as pomsets, which are the structures used in the tool PomCho [15] to check the closure conditions ensuring well-formedness of g-choreographies. However, A&A also challenges PomCho since the check of the closure conditions is implemented as a brute force algorithm. To overcome this issue we introduced optimisations of the algorithms and implemented them in PomCho. The verification of A&A initiates the usability analysis of PomCho advocated in [15].

The closure conditions proposed in [14] take inspiration from those in [1] and allow a more efficient analysis [15]. We extend the approach in [14] to synchronous interactions and adapt PomCho to support the new closure conditions. These results are instrumental to tackle the complexity of our case study.

The theories underlying PomCho and Corinne have common aims but different formalisations; we plan to investigate their relation in future work.

In [4] a mechanism to statically detect realisability in MSCs (a formal model of global specifications) has been proposed; the analysis is based on notions of non-local choices and of termination that are less permissive than our verification conditions since intra-participant concurrency is not allowed and termination awareness is not enforced.

Realisation of global views allowing interactions between multiple senders and multiple receivers is considered in [29]. Two notions of realisability are discussed, depending on whether local actions include the name of involved participants or not. Tool support is provided by the Ceta [7] tool.

Partial order models of choreographies akin to pomsets have been recently proposed in [6,10,11,9]. The pomset semantics of g-choreographies in [14,15] has inspired *branching pomsets* [10,11,9], which enable a more compact representation of choices. Whether this representation offers better algorithms for checking closure conditions is an open problem. Similar considerations apply to the results in [8], which proposes an optimisation of the pomset representation of concurrent computations.

The platform in [2] offers a framework for the top-down development of message-passing applications with low-level features (e.g., binding of components, or security aspects) but has limited functionalities for the analysis of global specifications.

As mentioned, our synchronous semantics and analysis build on the asynchronous ones. The reason was to reuse as much as possible previous PomCho implementation. One could instead represent synchronous communication as a single interaction event, labelled $\mathsf{A} \rightarrow \mathsf{B} : \mathsf{m}$. This enables to halve the number of events (obtaining a corresponding speedup), but requires an extensive rework of both the tool and the theory.

The current integrated approach could also help to exploit both the synchronisation mechanisms in a same model. This would allow to study heterogeneous systems. We leave this research direction for future work.

References

1. Rajeev Alur, Kousha Etessami, and Mihalis Yannakakis. Inference of Message Sequence Charts. *IEEE Trans. Software Eng.*, 29(7):623–633, 2003.

2. Marco Autili, Amleto Di Salle, Francesco Gallo, Claudio Pompilio, and Massimo Tivoli. Chorevolution: Automating the realization of highly-collaborative distributed applications. In *Coordination Models and Languages*, pages 92–108. Springer, 2019.
3. Franco Barbanera, Ivan Lanese, and Emilio Tuosto. Choreography automata. In Simon Bliudze and Laura Bocchi, editors, *Coordination Models and Languages - 22nd IFIP WG 6.1 International Conference, COORDINATION 2020, Held as Part of the 15th International Federated Conference on Distributed Computing Techniques, DisCoTec 2020, Valletta, Malta, June 15-19, 2020, Proceedings*, volume 12134 of *Lecture Notes in Computer Science*, pages 86–106. Springer, 2020.
4. Hanêne Ben-Abdallah and Stefan Leue. Syntactic detection of process divergence and non-local choice in message sequence charts. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 259–274. Springer, 1997.
5. Daniel Brand and Pitro Zafiropulo. On Communicating Finite-State Machines. *Journal of the ACM*, 30(2):323–342, 1983.
6. Ilaria Castellani, Mariangiola Dezani-Ciancaglini, and Paola Giannini. Event structure semantics for multiparty sessions. In *Models, Languages, and Tools for Concurrent and Distributed Programming - Essays Dedicated to Rocco De Nicola on the Occasion of His 65th Birthday*, volume 11665 of *Lecture Notes in Computer Science*, pages 340–363. Springer, 2019.
7. Ceta Tool. Available at `https://github.com/arcalab/choreo/tree/ceta`.
8. Luc Edixhoven. Shuffling posets on trajectories (technical report). *CoRR*, abs/2309.09189, 2023.
9. Luc Edixhoven and Sung-Shik Jongmans. Realisability of branching pomsets. In Silvia Lizeth Tapia Tarifa and José Proença, editors, *Formal Aspects of Component Software - 18th International Conference, FACS 2022, Virtual Event, November 10-11, 2022, Proceedings*, volume 13712 of *Lecture Notes in Computer Science*, pages 185–204. Springer, 2022.
10. Luc Edixhoven, Sung-Shik Jongmans, José Proença, and Ilaria Castellani. Branching pomsets: Design, expressiveness and applications to choreographies. *J. Log. Algebraic Methods Program.*, 136:100919, 2024.
11. Luc Edixhoven, Sung-Shik Jongmans, José Proença, and Guillermina Cledou. Branching pomsets for choreographies. In Clément Aubert, Cinzia Di Giusto, Larisa Safina, and Alceste Scalas, editors, *Proceedings 15th Interaction and Concurrency Experience, ICE 2022, Lucca, Italy, 17th June 2022*, volume 365 of *EPTCS*, pages 37–52, 2022.
12. Emmanuel Gaudin and Eric Brunel. Property Verification with MSC. In *International SDL Forum*, LNCS, pages 19–35. Springer, 2013.
13. Roberto Guanciale and E Tuosto. Realisability of pomsets via communicating automata. In *Proceedings 11th Interaction and Concurrency Experience, ICE*, pages 37–51, 2018.
14. Roberto Guanciale and Emilio Tuosto. Realisability of pomsets. *J. of Logic and Algebraic Methods in Programming*, 108:69–89, 2019.
15. Roberto Guanciale and Emilio Tuosto. Pomcho: a tool chain for choreographic design. *Science of Computer Programming*, 202:102535, 2021.
16. Healthcare Authorization and Accreditation protocol. Available at `https://salute.regione.emilia-romagna.it/ssr/strumenti-e-informazioni/autorizzazione-e-accreditamento/autorizzazione-e-accreditamento-sanitario`.
17. Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty Asynchronous Session Types. *Journal of the ACM*, 63(1):9:1–9:67, 2016.
18. Hans Hüttel, Ivan Lanese, Vasco T. Vasconcelos, Luís Caires, Marco Carbone, Pierre-Malo Deniélou, Dimitris Mostrous, Luca Padovani, António Ravara, Emilio Tuosto, Hugo Torres Vieira, and Gianluigi Zavattaro. Foundations of session types and behavioural contracts. *ACM Comput. Surv.*, 49(1):3:1–3:36, 2016.

19. Legge Regionale 06 Novembre 2019. Available at https://demetra.regione.emilia-romagna.it/al/articolo?urn=er:assemblealegislativa:legge:2019;22.
20. Formal description techniques (FDT) - Message Sequence Chart (MSC). Recommendation ITU-T Z.120, 2011. Available at http://www.itu.int/rec/T-REC-Z.120-201102-I/en.
21. Object Management Group. Business Process Model and Notation, 2011. http://www.bpmn.org.
22. Simone Orlando, Vairo Di Pasquale, Franco Barbanera, Ivan Lanese, and Emilio Tuosto. Corinne, a tool for choreography automata. In Gwen Salaün and Anton Wijs, editors, *Formal Aspects of Component Software - 17th International Conference, FACS 2021, Virtual Event, October 28-29, 2021, Proceedings*, volume 13077 of *Lecture Notes in Computer Science*, pages 82–92. Springer, 2021.
23. Sourabh Pal, Ivan Lanese, and Massimo Clo. Choreographic automata: A case study in healthcare management. In *COORDINATION*, volume 14676 of *Lecture Notes in Computer Science*, pages 3–19. Springer, 2024.
24. Sourabh Pal, Ivan Lanese, and Emilio Tuosto. Corinne-3. Available at https://github.com/lanese/corinne-3.
25. PomCho+ Tool repository. Available at https://bitbucket.org/sourabhphd/chorgram/src/ICTAC_2025/.
26. PomCho+ Tool repository: A&A rendered by PomCho+. Available at https://bitbucket.org/sourabhphd/chorgram/src/ICTAC_2025/ICTAC_Examples/Healthcare_Case_Study/choreography.png.
27. PomCho+ Tool repository: An example that satisfies CC3 but not CC2. Available at https://bitbucket.org/sourabhphd/chorgram/src/ICTAC_2025/ICTAC_Examples/CC2CC3/.
28. Vaughan Pratt. Modeling concurrency with partial orders. *International journal of parallel programming*, 15:33–71, 1986.
29. Maurice H ter Beek, Rolf Hennicker, and José Proença. Realisability of global models of interaction. In *International Colloquium on Theoretical Aspects of Computing*, pages 236–255. Springer, 2023.
30. Emilio Tuosto and Roberto Guanciale. Semantics of global view of choreographies. *J. of Logic and Algebraic Methods in Programming*, 95:17 – 40, 2018.