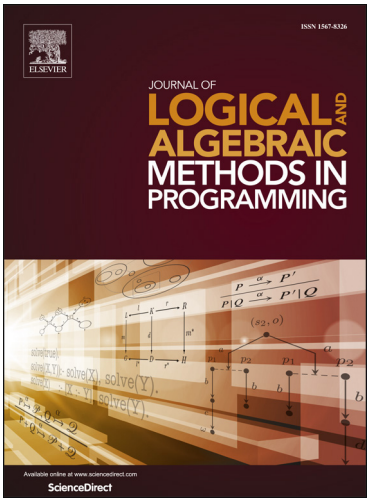
Accepted Manuscript



Realisability of pomsets

Roberto Guanciale, Emilio Tuosto

PII: S2352-2208(18)30166-4
DOI: https://doi.org/10.1016/j.jlamp.2019.06.003
Reference: JLAMP 470

To appear in: *Journal of Logical and Algebraic Methods in Programming*

Received date: 15 December 2018
Revised date: 8 May 2019
Accepted date: 19 June 2019

Please cite this article in press as: R. Guanciale, E. Tuosto, Realisability of pomsets, *J. Log. Algebraic Methods Program.* (2019), https://doi.org/10.1016/j.jlamp.2019.06.003

This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.

ACCEPTED MANUSCRIPT

Realisability of Pomsets☆

Roberto Guanciale[a,*], Emilio Tuosto[b,**]

[a]KTH Royal Institute of Technology, Sweden
[b]Gran Sasso Science Institute (IT) and Department of Informatics, University of Leicester (UK)

Abstract

Pomsets are a model of concurrent computations introduced by Pratt. We adopt pomsets as a syntax-oblivious specification model of distributed systems where coordination happens via asynchronous message-passing.

In this paper, we study conditions that ensure a specification expressed as a set of pomsets can be faithfully realised via communicating automata. Our main contributions are (i) the definition of a realisability condition accounting for termination soundness, (ii) conditions accounting for "multi-threaded" participants, and (iii) an algorithm to check our realisability conditions directly over pomsets, (iv) an analysis of the algorithm and its benchmarking attained with a prototype implementation.

1. Introduction

Problem. Distributed software is notoriously hard to design and implement. The complexity emerges from several factors. Unlike sequential software, modularisation helps only partially to tame the complexity of the problem. This is due to the fact that, no matter how one decomposes the problem at hand, by necessity there are multiple points of control. Execution and computational states are therefore scattered across multiple components. This makes it difficult to guarantee invariants of the computation: on the one hand, such invariants are properties of the *global* state emerging from the *local* states of the components; onye the other hand, design principles suggest to avoid centralisation points in order to reduce bottlenecks and increase scalability (see [21]) and robustness (limiting single points of failures).

Hence, distributed components have to coordinate with each other in order to "agree" on a global state to maintain invariants. The equation

$$\text{distributed software} \quad = \quad \text{distributed control} \quad + \quad \text{coordination}$$

helps to picture the consequences of this extra layer of complexity: for the satisfaction of the invariant it is crucial to attain correct information flows through components. One

☆ Research partly supported by the EU H2020 RISE programme under the Marie Skłodowska-Curie grant agreement No 778233.
The authors are grateful to the reviewers for the helpful comments.
*Corresponding author
**Corresponding author

ACCEPTED MANUSCRIPT

needs to develop components' coordination so to provide them with the information necessary to reach local states forming a global one satisfying the invariant. This is the focus of this paper. In particular we are concerned with the *realisability* of coordination.

We position ourselves in the context of *scenario-based* specifications of distributed software. A paradigmatic languages for scenario-based specifications are *message-sequence charts* (MSCs) [25]). In this context, architects specify distributed coordination of their application by providing a finite set of (finite) MSCs, that is *scenarios* that they want the application to guarantee. Basically, scenarios can be seen as global specifications of the desired coordination. As observed in [1], a source of problems is that there could be some specifications that are impossible to implement using the local views in a given communication model. The problem becomes evident when considering the schism [12, 31] between the design and the implementation level. In this paper we capitalize on the lesson learned in [31], in particular that by expressing conditions directly at the semantic level we can identify realizability notions that are not language specific. As we show in the paper our generalization of [1] can be applied to different choreographic formalisms. At design level, it is convenient to assume synchronous communications among components because it makes it easier to reason about coordination protocols. At the implementation level however, asynchronous communication is typically assumed. For instance, *asynchronous message-passing* is widely adopted in specification languages such as WS-CDL [32]) and programming languages such as Erlang, Scala, and Go and featured in message-oriented middlewares [23]. This communication paradigm is also at the core of several formal models (e.g. π-calculus [30, 24] and communicating automata [6]), choreography languages (e.g. global calculus [7].

Contributions. We study the realisability problem of scenario-based specifications given in terms of *partially ordered multisets* (pomsets) [28]. We propose pomsets as an abstract model for global scenarios to analyse their realisability in terms of asynchronous message-passing. More precisely, we assume asynchronous point-to-point communications with a notion of realisability that

1. rules out systems where some participants cannot ascertain termination
2. admits multi-threaded participants
3. allows us to define syntax-oblivious conditions
4. can be decided by an analysis of the partial orders of communication events.

These features have several practical advantages. Indeed, by (1), we admit systems where participants may get stuck on some messages, provided that the specification accounts for that. The use of multi-threaded participants (2) makes our framework more expressive than existing ones (see discussion on this point in [31]). Syntax independent conditions (3) are applicable to different global models. Finally, (4) enables the identification of design errors in global models rather than in execution traces where they are harder to analyse.

The pomset framework of this paper and its relations with [1] appeared in [13]. Besides improving the presentation of the results in [13], this paper

- provides a deeper discussion on the merits of our approach,

ACCEPTED MANUSCRIPT

- applies them in a choreographic context,
- gives examples to discuss the realisability of pomsets,
- provides a prototype implementation of our verification conditions discussing its computational complexity and showing its effectiveness in practical scenarios.

A positive by-product of our approach is the efficiency gain in the verification of the realisability conditions obtained when restricting to specific classes of choreographies characterisable in term of behavioural types.

Outline. Section 2 reviews basic definitions while Section 3 presents a language for global views and its pomset semantics. Section 4 discusses some elucidatory examples. Section 5 illustrates the problems of realisability and sound termination; there, we also recall (and adapt to pomsets) the verification conditions of [1]. Section 6 presents the sufficient conditions for realisability and sound termination that can be checked over partial orders. Section 7 describes an algorithm to check our realisability conditions, a prototype implementation, and shows an application to a realistic scenario. Section 8 discusses on the new verification conditions from a software engendering point of view. Finally, Section 9 discusses related work and Section 10 draws some conclusions.

2. Pomsets and message-sequence charts

We collect the main definitions needed in the rest of the paper. The material of this section is not an original contribution[1] and it is presented only to make the paper self-contained borrowing and combining definitions and notations from [10, 1, 18, 6].

Definition 1 (Lposets [10])**.** *A labelled partially-ordered set (lposet) is a triple* $(\mathcal{E}, \leq, \lambda)$*, with* $\mathcal{E}$ *a set of events,* $\leq \subseteq \mathcal{E} \times \mathcal{E}$ *a reflexive, anti-symmetric, and transitive relation on* $\mathcal{E}$*, and* $\lambda : \mathcal{E} \to \mathcal{L}$ *a labelling function mapping events in* $\mathcal{E}$ *to labels in* $\mathcal{L}$*.*

Intuitively, $\leq$ represents causality; for $e \neq e'$, if $e \leq e'$ and both events occur then e' is caused by e. Note that λ is not required to be injective: for $e \neq e' \in \mathcal{E}$, $\lambda(e) = \lambda(e')$ means that e and e' model different occurrences of the same action.

Definition 2 (Pomsets [10])**.** *Two lposets* $(\mathcal{E}, \leq, \lambda)$ *and* $(\mathcal{E}', \leq', \lambda')$ *are* isomorphic *if there is a bijection* $\phi : \mathcal{E} \to \mathcal{E}'$ *such that* $e \leq e' \iff \phi(e) \leq' \phi(e')$ *and* $\lambda = \lambda' \circ \phi$*. A partially-ordered multi-set (of actions), pomset for short, is an isomorphism class of lposets.*

Replacing lposets with pomsets allows us to abstract away from the names of events in $\mathcal{E}$. In the following, $[\mathcal{E}, \leq, \lambda]$ denotes the isomorphism class of $(\mathcal{E}, \leq, \lambda)$, symbols $r, r', \ldots$ (resp. $R, R', \ldots$) range over (resp. sets of) pomsets, and we assume that pomsets r contain at least one lposet which will possibly be referred to as $(\mathcal{E}_r, \leq_r, \lambda_r)$. An event e is an *immediate predecessor* of an event e' (or equivalently e' is an *immediate successor*

[1] Except for the different definition of accepting states of communicating automata.

ACCEPTED MANUSCRIPT

of e) in a pomset r if $e \neq e'$, $e \leq_r e'$, and for all $e'' \in \mathcal{E}_r$ such that $e \leq_r e'' \leq_r e'$ either $e = e''$ or $e' = e''$.

Hereafter, we consider pomsets labelled by communications representing output and input actions between a sender and a receiver. This is done by instantiating the set $\mathcal{L}$ of labels as follows.

Let $\mathcal{P}$ be a set of *participants* (ranged over by A, B, etc.), $\mathcal{M}$ a set (of types) of *messages* (ranged over by m, x, etc.). We take $\mathcal{P}$ and $\mathcal{M}$ disjoint. Participants coordinate with each other by exchanging messages over *communication channels*, that are elements of the set $\mathcal{C} = (\mathcal{P} \times \mathcal{P}) \setminus \{(\mathsf{A},\mathsf{A}) \mid \mathsf{A} \in \mathcal{P}\}$. We abbreviate $(\mathsf{A},\mathsf{B}) \in \mathcal{C}$ as AB. The set of *(communication) labels* $\mathcal{L}$ is defined by

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

The elements of $\mathcal{L}^!$ and $\mathcal{L}^?$, outputs and inputs, respectively represent *sending* and *receiving* actions; we shorten $(\mathsf{AB},!,\mathsf{m})$ as $\mathsf{AB!m}$ and $(\mathsf{AB},?,\mathsf{m})$ as $\mathsf{AB?m}$ and let l, l', ... range over $\mathcal{L}$. The *subject* of an action is defined by

$$\mathsf{sbj}(\mathsf{AB!m}) = \mathsf{A} \quad \text{(the sender)} \qquad \text{and} \qquad \mathsf{sbj}(\mathsf{AB?m}) = \mathsf{B} \quad \text{(the receiver)}$$

We will represent pomsets as (a variant[2] of) Hasse diagrams of the immediate predecessor relation; for instance, the pomset $r_{(1)} = [\{e_1,\ldots,e_4\}, \leq, \lambda]$ where $\lambda = \begin{cases} e_1 \mapsto \mathsf{AB!x} \\ e_2 \mapsto \mathsf{AB?x} \\ e_3 \mapsto \mathsf{AB!y} \\ e_4 \mapsto \mathsf{AB?y} \end{cases}$ and $\leq$ is the order induced by the immediate-successor represented by the edges in the following diagram

$$r_{(1)} = \left[\begin{array}{ccc} \mathsf{AB!x} & \longrightarrow & \mathsf{AB?x} \\ \downarrow & & \downarrow \\ \mathsf{AB!y} & \longrightarrow & \mathsf{AB?y} \end{array}\right] \tag{1}$$

The *projection* $r\!\downarrow_{\mathsf{A}}$ *of a pomset* r *on a participant* $\mathsf{A} \in \mathcal{P}$ is obtained by restricting r to the events having subject A: formally

$$r\!\downarrow_{\mathsf{A}} \quad = \quad [\mathcal{E}_{r,\mathsf{A}},\ \leq_r \cap\, (\mathcal{E}_{r,\mathsf{A}} \times \mathcal{E}_{r,\mathsf{A}}),\ \lambda_r|_{\mathcal{E}_{r,\mathsf{A}}}]$$

where $\mathcal{E}_{r,\mathsf{A}} = \{e \in \mathcal{E}_r \mid \mathsf{sbj}(\lambda_r(e)) = \mathsf{A}\}$.

Pomsets are a quite expressive model and encompass MSCs[3] which can be defined as a proper subclass of pomsets.

Definition 3 (Well-formedness, completeness, and MSCs)**.** *A pomset r over $\mathcal{L}$ is* well-formed *if for every event $e \in \mathcal{E}_r$*

1. *if $\lambda_r(e) = \mathsf{AB!m}$, there is at most one $e' \in \mathcal{E}_r$ immediate successor of e in r with $\lambda_r(e') = \mathsf{AB?m}$ (and, if such e' exists, we say that e and e'* match *each other)*

[2] Edges of Hasse diagrams are usually not oriented; here we use arrows so to draw order relations between events also horizontally.

[3] Pomsets can also be used to give semantics to the composition of MSCs; see [18].

ACCEPTED MANUSCRIPT

2. *if* $\lambda_r(e) = \mathsf{AB?m}$, *there exists exactly one* $e' \in \mathcal{E}_r$ *immediate predecessor of* e *in* r *with* $\lambda_r(e') = \mathsf{AB!m}$
3. *for each* $e' \in \mathcal{E}_r$, *if* e *is an immediate predecessor of* e' *and* $\mathit{sbj}(\lambda_r(e)) \neq \mathit{sbj}(\lambda_r(e'))$ *then* e *and* e' *are matching output and input events respectively*
4. *for each* $e' \neq e \in \mathcal{E}_r$ *with* $\lambda_r(e) = \lambda_r(e') = \mathsf{AB!m}$, *and for all* $\bar{e}, \bar{e}' \in \mathcal{E}_r$ *immediate successors in* r *of* e *and of* e' *respectively if* $\lambda_r(\bar{e}) = \lambda_r(\bar{e}') = \mathsf{AB?m}$ *and* $e \leq_r e'$ *then* $\bar{e}' \not\leq_r \bar{e}$

Pomset r *is* complete *if there is no send event in* $\mathcal{E}_r$ *without a matching receive event.*

A message-sequence chart *is a well-formed and complete pomset* r *such that* $\leq_{r\downarrow \mathsf{A}}$ *is a total order, for every* $\mathsf{A} \in \mathcal{P}$.

All conditions of Definition 3 are straightforward but condition (4), which requires that ordered output events with the same label cannot be matched by inputs that have opposite order.

Well-formed pomsets permit to represent inter-participant concurrency since they keep independent non matching communication events of different participants. MSCs are obtained by restricting participants to be single-threaded. The pomset $r_{(1)}$ indeed corresponds to an MSC. Vertical arrows represent orders on the events of a participant; for instance, the leftmost vertical arrow of $r_{(1)}$ represents that the output of the message x of A to B precedes the output of y. (In MSCs' jargon, this vertical arrow corresponds to the *lifeline* of A.) Likewise, the rightmost vertical arrow constrains the order of message reception. Basically, vertical arrows correspond to the *projections* of the pomsets on participants; these projections are obtained by restricting $r_{(1)}$ to the events having the same subject.

Well-formed pomsets can express intra-participant concurrency (i.e. multi-threaded participants) since they do not require $\leq_{r\downarrow \mathsf{A}}$ to be totally ordered. For example

$$r_{(2)} = \left[\begin{array}{ccc} \mathsf{AB!x} & \longrightarrow & \mathsf{AB?x} \\ \downarrow & & \\ \mathsf{AB!y} & \longrightarrow & \mathsf{AB?y} \end{array}\right] \tag{2}$$

is similar to $r_{(1)}$, but for the fact that it allows B to receive messages in any order. Note that $r_{(2)}$ cannot be expressed with MSCs exactly because the events of B are not totally ordered.

One may wonder why we use pomsets instead of simple partial orders. As shown in the next example, sometimes we need to impose order on multiple occurrences of the same communication events. For instance, in

$$r_{(3)} = \left[\begin{array}{ccc} \mathsf{AB!x} & \longrightarrow & \mathsf{AB?x} \\ \downarrow & & \downarrow \\ \mathsf{AB!x} & \longrightarrow & \mathsf{AB?x} \end{array}\right] \tag{3}$$

A and B exchange the same message twice in a row.

Distributed choices are modeled via sets of pomsets R, so that each $r \in R$ yields the causal dependencies of the communications in a branch. For instance,

$$R_{(4)} = \left\{ \left[\begin{array}{c} \mathsf{AB!x} \\ \downarrow \\ \mathsf{AB?x} \end{array}\right] \quad , \quad \left[\begin{array}{c} \mathsf{AB!y} \\ \downarrow \\ \mathsf{AB?y} \end{array}\right] \right\} \tag{4}$$

ACCEPTED MANUSCRIPT

represents a choice between exchanging message x or message y. As a further example,

$$R_{(5)} = \left\{ \left[\begin{array}{cc} \mathsf{AB!x} & \longrightarrow \mathsf{AB?x} \\ \downarrow & \downarrow \\ \mathsf{AB!y} & \longrightarrow \mathsf{AB?y} \end{array} \right] \quad , \quad \left[\begin{array}{cc} \mathsf{AB!x} & \longrightarrow \mathsf{AB?x} \\ \uparrow & \uparrow \\ \mathsf{AB!y} & \longrightarrow \mathsf{AB?y} \end{array} \right] \right\} \tag{5}$$

represents the fact that messages x and y can be exchanged in any order, but outputs and inputs must have the same order.

A natural question to ask is:

> "is it possible to realise $R_{(4)}$ and $R_{(5)}$ with asynchronously communicating local views? "

A similar question was answered for MSCs in [1]. Before answering the question in the more general case of pomsets (cf. Section 5) we review how global views of choreographies express more conveniently pomset-based scenarios. We remark that the conditions of Section 5 are oblivious of the coordination language; the model of the next section helps to illustrate how these can help in the design of distributed applications.

3. A choreographic model

Choreographic approaches have been advocated as suitable methodologies to handle the complexity of distributed systems [19]. These frameworks envisage two views: a global specification and a local one. The global view of a choreography is a contract containing the common ordering conditions and constraints under which messages are exchanged. The global specification is in turn realised by combination of the local systems, which defines the behavior of each participant.

Here we model the local systems in terms of *communicating automata*, which are formally introduced in Section 3.1. Pomsets can be seen as a model of global views of choreographies and yield an abstract semantic framework for them [31].

In order to provide a more abstract and convenient framework for the specification of scenarios closer to languages used in practice such as BPMN [27], Section 3.2 introduces a choreographic model for scenario-based specifications. We use this choreographic model to give an intuitive presentation of the examples in Section 4, while in Section 7 we use this model to discuss the effectiveness of our verification framework in this more practical contexts.

We simplify the framework in [31] by providing a pomset semantics of global specifications which does not account for their well-formedness. In other words, we move the verification of realisability from the analysis of the global specifications to the scenarios that they specify.

3.1. Local views

Local views are often conveniently modelled in terms of *communicating automata* of some sort. An A-*communicating finite state machine (*A-*CFSM)* $M = (Q, q_0, F, \rightarrow)$ is a finite-state automaton on the alphabet $\mathcal{L}$ such that, $q_0 \in Q$ is the initial state, $F \subseteq Q$ are the accepting states, and for each $q \xrightarrow{l} q'$ holds $\mathsf{sbj}(l) = \mathsf{A}$. A *(communicating)*

ACCEPTED MANUSCRIPT

system is a map $S = (M_{\mathsf{A}})_{\mathsf{A}\in\mathcal{P}}$ assigning an A-CFSM M_{A} to each participant $\mathsf{A} \in \mathcal{P}$. For all $\mathsf{A} \neq \mathsf{B} \in \mathcal{P}$, we shall use an unbounded multiset b_{AB} where M_{A} puts the message to M_{B} and from which M_{B} consumes the messages from M_{A}.

The semantics of communicating systems is defined in terms of transition relations between *configurations* which keep track of the state of each machine and the content of each bag (i.e. unordered buffer) . Let $S = (M_{\mathsf{A}})_{\mathsf{A}\in\mathcal{P}}$ be a communicating system. A *configuration* of S is a pair $s = \langle \vec{q} \,;\, \vec{b} \rangle$ where $\vec{q} = (q_{\mathsf{A}})_{\mathsf{A}\in\mathcal{P}}$ maps each participant A to its local state $q_{\mathsf{A}} \in Q_{\mathsf{A}}$ and $\vec{b} = (b_{\mathsf{AB}})_{\mathsf{AB}\in\mathcal{C}}$ where the bag $b_{\mathsf{AB}} : \mathcal{M} \to \mathbb{N}$ is a map assigning the number of occurrences of each message; state q_{A} keeps track of the state of the automaton M_{A} and bag b_{AB} keeps track of the messages sent from A to B. The *initial* configuration s_0 is the one where, for all $\mathsf{A} \in \mathcal{P}$, q_{A} is the initial state of the corresponding CFSM and all bags are empty. Given two configurations $s = \langle \vec{q} \,;\, \vec{b} \rangle$ and $s' = \langle \vec{q}' \,;\, \vec{b}' \rangle$, relation $s \overset{l}{\Rightarrow} s'$ holds if there is a message $\mathsf{m} \in \mathcal{M}$ such that either (1) or (2) below holds:

1. $l = \mathsf{AB!m}$ and $q_{\mathsf{A}} \xrightarrow{l}_{\mathsf{A}} q'_{\mathsf{A}}$ and
   a. $q'_{\mathsf{C}} = q_{\mathsf{C}}$ for all $\mathsf{C} \neq \mathsf{A} \in \mathcal{P}$,
   b. $b'_{\mathsf{AB}} = b_{\mathsf{AB}}[\mathsf{m} \mapsto b_{\mathsf{AB}}(\mathsf{m}) + 1]$,
   c. $b'_{\mathsf{CD}} = b_{\mathsf{CD}}$ for all $\mathsf{CD} \neq \mathsf{AB}$
2. $l = \mathsf{AB?m}$ and $q_{\mathsf{B}} \xrightarrow{l}_{\mathsf{B}} q'_{\mathsf{B}}$ and
   a. $q'_{\mathsf{C}} = q_{\mathsf{C}}$ for all $\mathsf{C} \neq \mathsf{B} \in \mathcal{P}$,
   b. $b_{\mathsf{AB}}(\mathsf{m}) > 0$, $b'_{\mathsf{AB}} = b_{\mathsf{AB}}[\mathsf{m} \mapsto b_{\mathsf{AB}}(\mathsf{m}) - 1]$,
   c. $b'_{\mathsf{CD}} = b_{\mathsf{CD}}$ for all $\mathsf{CD} \neq \mathsf{AB}$

where, $f[x \mapsto y]$ is the usual notation for the updating of a function f in a point x of its domain with a value y. Condition (1) puts m on channel AB, while (2) gets m from channel AB by simply updating the number of occurrences of m in the bag b_{AB}. In both cases, any machine or bag not involved in the transition is left unchanged in the new configuration s'.

The automata model adopted in [1] is a slight variant of *communicating-finite state machines* (CFSMs) [6]. The two models have the same definition of automata; they differ in how communication is attained, but are equivalent up to internal transitions (which in [1] have been used to simplify proofs). We used the definition of CFSMs in [6] to encompass accepting states (necessary to define our notion of termination soundness Definition 6). Another deviation from the definition of CFSMs introduced in [6] is that bags become multisets in [1] while in [6] they follow a FIFO policy.
In order to avoid representing intra-participant concurrency with explicit interleaving a more abstract model than CFSMs (such as session types or Petri nets) could have been used for local views. However, since CFSMs are Turing complete (and therefore they can express all possible local behaviors in the given communication model), we consider them as a yardstick for realisability. Moreover, CFSMs feature a communication model very similar to the ones used in several programming languages or communication middlewares, like Erlang, Scala, and Akka.

Given a communicating system S, a configuration $s = \langle \vec{q} \,;\, \vec{b} \rangle$ of S is (*i*) *accepting* if all buffers in $\vec{b}$ are empty and the local state $\vec{q}(\mathsf{A})$ of each participant A is accepting while (*ii*) s is a *deadlock* TODO if no accepting configuration is reachable from s. We can then define the *language of* S as the set $\mathbb{L}(S) \in \mathcal{L}^\star$ of sequences $l_0 \dots l_{n-1}$ such that $s_0 \overset{l_0}{\Rightarrow} \dots \overset{l_{n-1}}{\Rightarrow} s_n$ and s_n is an accepting configuration.

We adopt the definition of deadlock given in [**?**]. Notice that this definition is

ACCEPTED MANUSCRIPT

intended to characterize systems that cannot terminate due to coordination problems and differ with respect to the notion of deadlock in [8], which characterizes systems that are not live.

3.2. Global views

Since scenario based approaches only consider finite protocols, we adopt the loop-free fragment of the design language proposed in [9] and extended in [31, 12] where a pomset semantics is also provided. In our language, dubbed *global choreography* (g-choreography for short), *interactions* are the units of coordination. G-choreographies are expressed according to the following grammar:

$$\mathsf{G} ::= \mathbf{0} \mid \mathsf{A}{\to}\mathsf{B}\colon \mathsf{m} \mid \mathsf{G};\mathsf{G}' \mid \mathsf{G} \mid \mathsf{G}' \mid \mathsf{G} + \mathsf{G}'$$

A g-choreography can be empty; a simple interaction that represents the fact that participant A sends message m to participant B, which is expected to receive m; sequential and parallel composition of g-choreographies; or the non-deterministic choice between two g-choreographies. We implicitly assume $\mathsf{A} \neq \mathsf{B}$ in interactions $\mathsf{A}{\to}\mathsf{B}\colon \mathsf{m}$. Each g-choreography G can be represented as a graph. Akin to BPMN [27] diagrams, the graphical notation in Fig. 1 yields a visual description of g-choreographies.

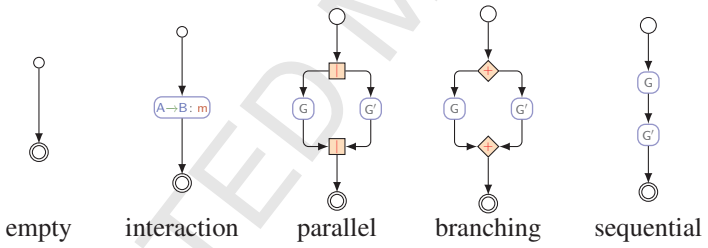


Figure 1: Our graphs: ○ is the source node, ◎ the sink one

The semantics of a choice-free g-choreography G (i.e. a choreography that does not contain $\_ + \_$ terms) can be expressed using a pomset, which represents the causal dependencies of the communication actions specified by G. Intuitively, the semantics of $\mathsf{G} + \mathsf{G}'$ consists of two (sets of) pomsets, one representing the causal dependencies of the communication actions of G and the other of those of G'.

In order to define a pomset-based semantics of choreographies, we define two important constructions to compose pomsets in parallel and sequentially, and corresponding notations. In the following, given a natural number n, $\mathbf{n}$ represents the singleton $\{n\}$. Also, we use $X \uplus Y$ to represent the disjoint union of two sets X and Y: $X \uplus Y = (X \times \mathbf{1}) \cup (Y \times \mathbf{2})$. Finally, given a function f on X, we define $f \otimes \mathbf{n} = \{(x,n) \mapsto f(x) \mid x \in X\}$ as the function extending f to $X \times \mathbf{n}$; analogously, for a relation $R \subseteq X \times Y$, we let $R \otimes \mathbf{n} = \{((x,n),(y,n)) \mid (x,y) \in R\}$ be the relation extending R to $(X \times \mathbf{n}) \times (Y \times \mathbf{n})$.

ACCEPTED MANUSCRIPT

Definition 4. *Let* $r = [\mathcal{E}, \leq, \lambda]$ *and* $r' = [\mathcal{E}', \leq', \lambda']$ *be two pomsets. The* parallel composition of r and r' *is:*

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

The sequential composition of r and r' *is:*

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

where

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

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

Both parallel and sequential composition preserve the causal dependencies of its constituents $\leq$ and $\leq'$. However, while there is no new dependency introduced by the parallel composition, the sequential composition of two pomsets adds to those in $\leq$ and $\leq'$ the dependencies among events in r and r' with the same subject. Basically, a causal relation is induced whenever a participant performing a communication in r also performs a communication in r'.

The semantics of a g-choreography is a family of pomsets defined as

$$\begin{aligned}
[\![\mathbf{0}]\!] &= \{\varepsilon\} \\
[\![\mathsf{A}{\to}\mathsf{B}\colon \mathsf{m}]\!] &= \{[(\{e_1, e_2\}, \{(e_1, e_1), (e_2, e_2), (e_1, e_2)\}, \lambda)]\} \text{ where } \lambda : \begin{cases} e_1 \mapsto \mathsf{A}\,\mathsf{B}!\mathsf{m} \\ e_2 \mapsto \mathsf{A}\,\mathsf{B}?\mathsf{m} \end{cases} \\
[\![\mathsf{G} \mid \mathsf{G}']\!] &= \{\mathsf{par}(r, r') \mid (r, r') \in [\![\mathsf{G}]\!] \times [\![\mathsf{G}']\!]\} \\
[\![\mathsf{G}; \mathsf{G}']\!] &= \{\mathsf{seq}(r, r') \mid (r, r') \in [\![\mathsf{G}]\!] \times [\![\mathsf{G}']\!]\} \\
[\![\mathsf{G} + \mathsf{G}']\!] &= [\![\mathsf{G}]\!] \cup [\![\mathsf{G}']\!]
\end{aligned}$$

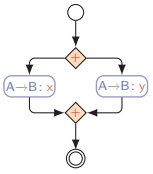
The semantics of the empty g-choreography $\mathbf{0}$ and of interaction $\mathsf{A}{\to}\mathsf{B}\colon \mathsf{m}$ are straightforward; for the latter, the send part $\mathsf{A}\,\mathsf{B}!\mathsf{m}$ of the interaction must precede its receive part $\mathsf{A}\,\mathsf{B}?\mathsf{m}$. For the parallel composition $\mathsf{G} \mid \mathsf{G}'$ we take the union of the dependencies of every possible execution, thus allowing the concurrent occurrence of the events of each thread. The semantics of sequential composition $\mathsf{G}; \mathsf{G}'$ establishes happens-before relations as computed by $\mathsf{seq}(r, r')$. Finally, the semantics of choice enables all pomsets of the constituent branches.

4. Realisability problems by examples

We now consider a few examples to discuss and anticipate some of the problems related to implementing pomset-based scenarios using concurrent agents. In the examples we use the visual representation of g-choreographies together with the corresponding pomset semantics.

ACCEPTED MANUSCRIPT

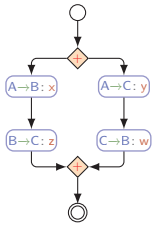
We first turn our attention to non-deterministic g-choreographies. Consider the simple choice $\mathsf{G}_{(4)} = \mathsf{A}\to\mathsf{B}\colon \mathsf{x} + \mathsf{A}\to\mathsf{B}\colon \mathsf{y}$ depicted below



$$[\![\mathsf{G}_{(4)}]\!] = \left\{ \begin{bmatrix} \mathsf{AB!x} \\ \downarrow \\ \mathsf{AB?x} \end{bmatrix} , \begin{bmatrix} \mathsf{AB!y} \\ \downarrow \\ \mathsf{AB?y} \end{bmatrix} \right\}$$

The semantics $[\![\mathsf{G}_{(4)}]\!]$ contains one pomset for each branch. It is easy to realise $\mathsf{G}_{(4)}$ since participant A decides which message (between x and y) is delivered to B.

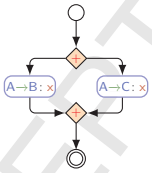
Another example of choreography that can be implemented by distributed components is $\mathsf{G}_{(6)} = \mathsf{A}\to\mathsf{B}\colon \mathsf{x};\mathsf{B}\to\mathsf{C}\colon \mathsf{z} + \mathsf{A}\to\mathsf{C}\colon \mathsf{y};\mathsf{C}\to\mathsf{B}\colon \mathsf{w}$ for which we have



$$[\![\mathsf{G}_{(6)}]\!] = \left\{ \begin{bmatrix} \mathsf{AB!x} \\ \downarrow \\ \mathsf{AB?x} \\ \downarrow \\ \mathsf{BC!z} \\ \downarrow \\ \mathsf{BC?z} \end{bmatrix} , \begin{bmatrix} \mathsf{AC!y} \\ \downarrow \\ \mathsf{AC?y} \\ \downarrow \\ \mathsf{CB!w} \\ \downarrow \\ \mathsf{CB?w} \end{bmatrix} \right\} \tag{6}$$

Here, participant A decides which branch is taken and informs either B or C, which in turn notifies the third participant. Intuitively, participant B can wait for a message coming from A or C and identify the selected branch accordingly.

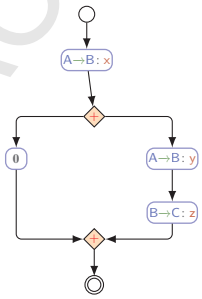
The choreography $\mathsf{G}_{(7)} = \mathsf{A}\to\mathsf{B}\colon \mathsf{x} + \mathsf{A}\to\mathsf{C}\colon \mathsf{x}$ provides an example of problematic choices. We have



$$[\![\mathsf{G}_{(7)}]\!] = \left\{ \begin{bmatrix} \mathsf{AB!x} \\ \downarrow \\ \mathsf{AB?x} \end{bmatrix} , \begin{bmatrix} \mathsf{AC!x} \\ \downarrow \\ \mathsf{AC?x} \end{bmatrix} \right\} \tag{7}$$

Here A decides to which participant the message x is delivered. The participant not selected by A has no way to identify if/when the choice has been made.

Consider now $\mathsf{G}_{(8)} = \mathsf{A}\to\mathsf{B}\colon \mathsf{x};\big(\mathbf{0} + \mathsf{A}\to\mathsf{B}\colon \mathsf{y};\mathsf{B}\to\mathsf{C}\colon \mathsf{z}\big)$; we have

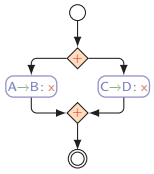


$$[\![\mathsf{G}_{(8)}]\!] = \left\{ \begin{bmatrix} \mathsf{AB!x} \\ \downarrow \\ \mathsf{AB?x} \end{bmatrix} , \begin{bmatrix} \mathsf{AB!x} & \longrightarrow & \mathsf{AB!x} \\ \downarrow & & \downarrow \\ \mathsf{AB?y} & \longrightarrow & \mathsf{AB?y} \\ & & \downarrow \\ & & \mathsf{BC!z} \\ & & \downarrow \\ & & \mathsf{BC?z} \end{bmatrix} \right\} \tag{8}$$

ACCEPTED MANUSCRIPT

The choice in $G_{(8)}$ is unsound. In fact, after the interaction $A \to B : x$, the choreography can either immediately terminate or continue with interactions $A \to B : y$ and $B \to C : z$. This can lead the participants B and C to wait indefinitely.

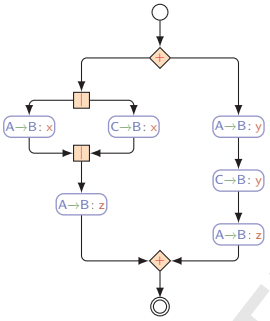
Another example of incorrect choice is $G_{(9)} = A \to B : x + C \to D : x$ represented below with its semantics:



$$[\![G_{(9)}]\!] = \left\{ \begin{bmatrix} AB!x \\ \downarrow \\ AB?x \end{bmatrix} , \begin{bmatrix} CD!x \\ \downarrow \\ CD?x \end{bmatrix} \right\} \tag{9}$$

Basically, $G_{(9)}$ requires both A and C to commit to a distributed choice without any communication among them.

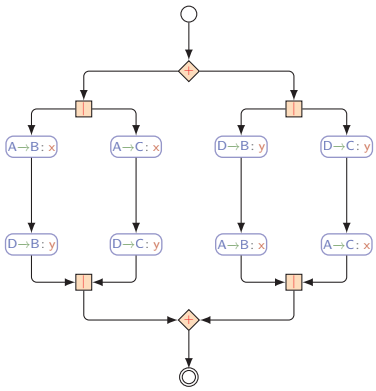
The choreography $G_{10} = G + G'$, where $G = (A \to B : x \mid C \to B : x); A \to B : z$ and $G' = A \to B : y; C \to B : y; A \to B : z$, has a similar obstacle for realisability. The graphical representation and the semantics of $G_{(10)}$ are



$$[\![G_{(10)}]\!] = \left\{ \begin{bmatrix} AB!x & \to & AB?x & & CB?x & \leftarrow & CB!x \\ \downarrow & & & \searrow & & \swarrow & \\ AB!z & & \to & & AB?z & & \end{bmatrix}, \begin{bmatrix} AB!y & \to & AB?y & & CB!y \\ \downarrow & & \downarrow & \swarrow & & & \\ AB!z & & CB?y & & & & \\ & \searrow & \downarrow & & & & \\ & & AB?z & & & & \end{bmatrix} \right\} \tag{10}$$

Here, participants A and C should both send the message x or both send the message y. However, A and C do not coordinate to achieve this behaviour; this makes it impossible for them to distributively commit to a common choice.

Let $G_{(11)} = G + G'$ where $G = (A \to B : m; D \to B : y) \mid (A \to C : x; D \to C : y)$ and $G' = (D \to B : y; A \to B : m) \mid (D \to C : y; A \to C : x)$. Pictorially $G_{(11)}s$ is



(11)

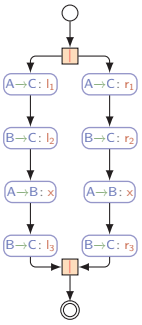
ACCEPTED MANUSCRIPT

has two branches that describe different orders of the same set of events; in fact

$$[\![\mathsf{G}_{(11)}]\!] = \left\{ \left[\begin{array}{ccc} \mathsf{AB!x} & \longrightarrow & \mathsf{AB?x} \\ & & \downarrow \\ \mathsf{DB?y} & \longleftarrow & \mathsf{DB!y} \\ \mathsf{AC!x} & \longrightarrow & \mathsf{AC?x} \\ & & \downarrow \\ \mathsf{DC?y} & \longleftarrow & \mathsf{DC!y} \end{array} \right], \left[\begin{array}{ccc} & & \mathsf{DB?y} \longleftarrow \mathsf{DB!y} \\ & & \downarrow \\ \mathsf{AB!x} & \longrightarrow & \mathsf{AB?x} \\ & & \mathsf{DC?y} \longleftarrow \mathsf{DC!y} \\ & & \downarrow \\ \mathsf{AC!x} & \longrightarrow & \mathsf{AC?x} \end{array} \right] \right\}$$

The behaviour of A (and D) is the same in both branches: A (resp. D) concurrently sends message x (resp. y) to B and C. The behaviours of B and C differ: in the left branch they first receive the message from A then the one from D, in the right branch, they have the same interactions but in opposite order. This choreography cannot be realised since, intuitively, it requires B and C to commit on the same order of reception without communicating with each other.

The presence of concurrent threads can introduce problems as well. The choreography $\mathsf{G}_{(12)} = \mathsf{G} \mid \mathsf{G}'$, where $\mathsf{G} = \mathsf{A}{\to}\mathsf{C}\colon \mathsf{l}_1; \mathsf{B}{\to}\mathsf{C}\colon \mathsf{l}_2; \mathsf{A}{\to}\mathsf{B}\colon \mathsf{x}; \mathsf{B}{\to}\mathsf{C}\colon \mathsf{l}_3$ and $\mathsf{G}' = \mathsf{A}{\to}\mathsf{C}\colon \mathsf{r}_1; \mathsf{B}{\to}\mathsf{C}\colon \mathsf{r}_2; \mathsf{A}{\to}\mathsf{B}\colon \mathsf{x}; \mathsf{B}{\to}\mathsf{C}\colon \mathsf{r}_3$, consists of two concurrent choreographies:



$$[\![\mathsf{G}_{(12)}]\!] = \left\{ \left[\begin{array}{ccc} \mathsf{AC!l_1} & \longrightarrow & \mathsf{AC?l_1} \\ & & \mathsf{BC!l_2} \longrightarrow \mathsf{BC?l_2} \\ & & \downarrow \\ \mathsf{AB!x} & \longrightarrow & \mathsf{AB?x} \\ & & \downarrow \\ & & \mathsf{BC!l_3} \longrightarrow \mathsf{BC?l_3} \end{array} \right], \left[\begin{array}{ccc} \mathsf{AC!r_1} & \longrightarrow & \mathsf{AC?r_1} \\ & & \mathsf{BC!r_2} \longrightarrow \mathsf{BC?r_2} \\ & & \downarrow \\ \mathsf{AB!x} & \longrightarrow & \mathsf{AB?x} \\ & & \downarrow \\ & & \mathsf{BC!r_3} \longrightarrow \mathsf{BC?r_3} \end{array} \right] \right\} \quad (12)$$

The semantics of this choreography consists of a single pomset, which is equal to the disjoint union of two pomsets. In this example, the usage of the same message in the two constituent threads can cause the following problem:

1. the left thread of A executes $\mathsf{AC!l_1}$ and $\mathsf{AB!x}$
2. after the output $\mathsf{BC!r_2}$, the right thread of B executes the input $\mathsf{AB?x}$, so "stealing" the message x generated by the left thread of A and meant for the left thread of B
3. the right thread of B executes $\mathsf{BC!r_3}$.

This violates the constraint that event $\mathsf{AC!r_1}$ must always precede event $\mathsf{BC!r_3}$, which the specification imposes independently of the interleaved execution of the participants' threads.

Realisability conditions that are syntax-oblivious permit to analyze cases that do not correspond to g-choreographies.

$$r_{13.a} = \left[\begin{array}{ccc} \mathsf{AB!x} & \longrightarrow & \mathsf{AB?x} \\ & & \downarrow \\ & & \mathsf{CB?x} \longleftarrow \mathsf{CB!x} \\ & & \downarrow \\ \mathsf{AB!y} & \longrightarrow & \mathsf{AB?y} \end{array} \right] \quad r_{13.b} = \left[\begin{array}{ccc} \mathsf{AB!z} & \longrightarrow & \mathsf{AB?z} \\ & & \downarrow \\ & & \mathsf{CB?z} \longleftarrow \mathsf{CB!z} \\ & & \downarrow \\ \mathsf{AB!y} & \longrightarrow & \mathsf{AB?y} \end{array} \right] \quad (13)$$

ACCEPTED MANUSCRIPT

In the choreography represented by $r_{13.a}$, A can send messages in any order, while B must first receive the message x. The singletons $\{r_{13.a}\}$ and $\{r_{13.b}\}$ are easily realisable using CFSMs. However there is no g-choreography whose semantics is exactly $\{r_{13.a}\}$ or $\{r_{13.b}\}$, because the participant B would have to be multi-threaded while in the semantics the events of B are sequentially ordered. Therefore, the set of pomsets $R_{13} = \{r_{13.a}, r_{13.b}\}$ is also not expressible as g-choreography. Note however, that the set R_{13} is not realisable, because it requires both A and C to commit to a distributed choice (delivery of message x or z) without any communication among them.

5. Realisability and termination soundness of pomsets

The notion of *realisability* and *sound termination* (cf. Definitions 5 and 6 below) are given in terms of the relation between the *language* of the global view and the one of a system of local views "implementing" it. Our notion of realisability considers languages over $\mathcal{L}$ as sets of traces of the distributed executions of the CFSMs discussed in Section 3.1, analogously to [1]. Hereafter, we assume all structures, including languages, words and pomsets, to be finite.

Given a pomset r, a *linearization* of r is a string in $\mathcal{L}^\star$ obtained by considering a total ordering of the events $\mathcal{E}_r$ that is consistent with the partial order $\leq_r$, and then replacing each event by its label. More precisely, let $|\ \mathcal{E}_r\ |$ be the cardinality of $\mathcal{E}_r$, a word $w = \lambda_r(e_1)\ldots\lambda_r(e_{|\mathcal{E}_r|})$ is a *linearization* of a pomset r if $e_1 \ldots e_{|\mathcal{E}_r|}$ is a permutation that totally orders the events in $\mathcal{E}_r$ so that if $e_i \leq_r e_j$ then $i \leq j$. For a pomset r, define $\mathbb{L}(r)$ to be the set of all linearizations of r. A word w over $\mathcal{L}$ is *well-formed* (resp. *complete*) if it is the linearization of a well-formed (resp. *complete*) pomset. Hereafter, for a word $w \in \mathcal{L}^\star$, $w\downarrow_{\mathsf{A}}$ denotes the projection of w that retains only those events where participant $\mathsf{A} \in \mathcal{P}$ is the subject. Operation $\_\downarrow_{\mathsf{A}}$ acts element-wise on languages over $\mathcal{L}$. The *language* of a set of pomsets R is simply defined as $\mathbb{L}(R) = \bigcup_{r\in R}\mathbb{L}(r)$.

We can now give the notion of *realisability* .

Definition 5 (Realisability)**.** *A language $L \subseteq \mathcal{L}^\star$ is* weakly realisable *if there is a communicating system S such that $L = \mathbb{L}(S)$; when S is deadlock-free we say that L is* safely realisable*. A set of pomsets R is* weakly *(resp.* safely*)* realisable *if $\mathbb{L}(R)$ is weakly (resp. safely) realisable.*

The notion of realisability is meaningful when pomsets are *well-formed* and *complete*, namely when they yield a proper match among receive and send events.

In general, safe realisability is not enough to rule out undesirable designs. In fact, it admits systems where participants cannot ascertain termination and may be left waiting forever for some messages. This may lead non-terminating participants to unnecessarily lock resources once the coordination is completed. We explain this considering $\mathsf{G}_{(8)}$ which can be interpreted as follows. Participant A starts a transaction with B by sending message x. The left branch (i.e. left pomset of $[\![\mathsf{G}_{(8)}]\!]$) represents a scenario where the transaction was started but neither committed nor aborted. The right branch (i.e. the right pomset) represents a scenario where the transaction started and eventually committed. Yet, B is uncertain whether message y is going to be sent or not and hence B be could locally decide to terminate immediately after receiving x leaving C waiting

ACCEPTED MANUSCRIPT

for message z. However, depending on the application requirements, it may be the case that termination awareness is important for B and not for C because e.g., either C is not "wasting" resources or it is immaterial that such resources are left locked. To handle this limitation we introduce a novel termination condition, which allows to specify the subset of participants that should be able to identify when no further message can be exchanged.

The notion of *sound termination* requires that in accepting configurations some participants of interest do not have input transitions making them wait while other participants have terminated.

Definition 6 (Termination soundness)**.** *A participant* $\mathsf{A} \in \mathcal{P}$ *is* termination-unaware *in a system S if there exists an accepting configuration* $\langle \vec{q}\,;\,\vec{b}\rangle$ *reachable in S having a transition departing from* $\vec{q}(\mathsf{A})$ *that is labelled in* $\mathcal{L}^?$.

A set of participants $\mathcal{P}' \subseteq \mathcal{P}$ *is* termination-aware *in a system S if no participant* $\mathsf{A} \in \mathcal{P}'$ *is termination-unaware in S. A language L over* $\mathcal{L}$ *is* termination-sound *for* $\mathcal{P}' \subseteq \mathcal{P}$ *if L is safely realisable by a system for which* $\mathcal{P}'$ *is termination-aware. A set of pomsets R is* termination-sound *for* $\mathcal{P}'$ *if* $\mathbb{L}(R)$ *is termination-sound for* $\mathcal{P}'$.

Realisability and termination soundness can be established by analyzing verification conditions of the language. In [1] two closure conditions are introduced that entail weak and safe realisability. A word w over $\mathcal{L}$ is $\mathcal{P}$*-feasible* for $L \subseteq \mathcal{L}^\star$ if $\forall \mathsf{A} \in \mathcal{P} : \exists w' \in L : w\downarrow_{\mathsf{A}} = w'\downarrow_{\mathsf{A}}$. In [1], a language L over the alphabet $\mathcal{L}$ that enjoys the following condition

$$L \supseteq \{w \in \mathcal{L}^\star \mid w \text{ well-formed, complete, and } \mathcal{P}\text{-feasible for } L\}$$

is said[4] to be **CC2**. Intuitively, the closure condition **CC2** entails that L is realisable by the set of participants performing the actions in $\mathcal{L}$: if each participant cannot tell apart a trace w with one of its expected executions (i.e., those in L) then w must be in L or, in the terminology of [1], w is *implied*. Closure condition **CC2** characterises the class of weakly realisable languages over $\mathcal{L}$.

Theorem 1 ([1])**.** *A language L is weakly realisable if, and only if, L contains only well-formed and complete words and satisfies* ***CC2****.*

The language $\mathbb{L}(R_{(11)})$ of the set of pomsets $R_{(11)}$ of Eq. (11) is not closed under **CC2**. In fact, the well-formed and complete word

$$\mathsf{AB!x};\ \mathsf{AB?x};\ \mathsf{DB!y};\ \mathsf{DB?y};\ \mathsf{DC!y};\ \mathsf{DC?y};\ \mathsf{AC!x};\ \mathsf{AC?x} \tag{14}$$

satisfies the conditions of **CC2**, because the projection of the word (14) on each participant equals the projection of a linearization of one of the pomsets in $R_{(11)}$ on the same participant. However, (14) is not in the language $\mathbb{L}(R_{(11)})$, because AC?x must precede DC?y in all the words obtained by the linearization of the first pomset in $R_{(11)}$, while in those obtained by a linearization of the other pomset in $R_{(11)}$, DB?y must precede AB?x.

[4] We stick with the terminology in [1] where closure conditions are not given specific names.

ACCEPTED MANUSCRIPT

The realisability entailed by condition **CC2** is "weak" because it does not rule out possibly deadlocking systems. Therefore, an additional closure condition, dubbed **CC3**, has been identified in [22, 1]. A language L over the alphabet $\mathcal{L}$ has the closure condition **CC3** when

$$\mathsf{pref}(L) \supseteq \{w \in \mathcal{L}^\star \mid w \text{ well-formed and } \mathcal{P}\text{-feasible for } \mathsf{pref}(L)\}$$

where $\mathsf{pref}(L)$ is the prefix closure of L. Basically, condition **CC3** states that any (partial) execution that cannot be told apart by any of the participants is a (partial) execution in L. And now the following result characterises safe realisability.

Theorem 2 ([22, 1])**.** *A language L is safe realisable if, and only if, L contains only well-formed and complete words and satisfies **CC2** and **CC3** [5].*

Once a language L is known to be realisable, we get a system $S(L) = (M_{\mathsf{A}})_{\mathsf{A}\in\mathcal{P}}$ realising L by defining, for all $\mathsf{A} \in \mathcal{P}$

$$M_{\mathsf{A}} = (\mathsf{pref}(L\!\downarrow_{\mathsf{A}}), \varepsilon, L\!\downarrow_{\mathsf{A}}, \rightarrow) \quad \text{where } w \xrightarrow{l} w.l \text{ if } w.l \in \mathsf{pref}(L\!\downarrow_{\mathsf{A}})$$

Then, in [1] the following result is shown.

Theorem 3 ([1])**.** *If L is a weakly realisable language then $\mathbb{L}(S(L)) = L$. Moreover, if L is safely realisable then $S(L)$ is deadlock-free.*

We introduce a new verification condition for termination soundness. A participant $\mathsf{A} \in \mathcal{P}$ is *termination-unaware* for the language L over $\mathcal{L}$ if there exist $w, w' \in L$ such that $w\!\downarrow_{\mathsf{A}}$ is a prefix of $w'\!\downarrow_{\mathsf{A}}$ and the first symbol in $w'\!\downarrow_{\mathsf{A}}$ after $w\!\downarrow_{\mathsf{A}}$ is in $\mathcal{L}^?$. Given a set of participants $\mathcal{P}' \subseteq \mathcal{P}$, we say that L is *$\mathcal{P}'$-terminating* when there is no $\mathsf{A} \in \mathcal{P}'$ termination-unaware for L. The language of the family of pomsets $[\![\mathsf{G}_{(8)}]\!]$ is $\{\mathsf{A}\}$-terminating. However, such language is not $\{\mathsf{B}\}$-terminating. In fact, after receiving the message $\mathsf{AB?x}$, participant B cannot distinguish whether A terminates or will send $\mathsf{AB!y}$; hence B ends up in a state where it is ready to fire the input $\mathsf{AB?y}$, but no matching output could arrive from A. And likewise for C.

Theorem 4. *For $\mathcal{P}' \subseteq \mathcal{P}$, if L is $\mathcal{P}'$-terminating and safely realisable then it is termination-sound for $\mathcal{P}'$.*

Proof. The proof is straightforward. Let $S(L)$ be the system obtained from the construction of Theorem 3. $S(L)$ is deadlock-free and $L = \mathbb{L}(S(L))$. Let $\mathsf{A} \in \mathcal{P}'$, $w \in L$, and s an accepting configuration reached in a run of S corresponding to w. For each $w' \in L$ such that $w\!\downarrow_{\mathsf{A}}$ is prefix of $w'\!\downarrow_{\mathsf{A}}$, the first symbol in $w'\!\downarrow_{\mathsf{A}}$ after $w\!\downarrow_{\mathsf{A}}$ cannot be an input (since L is $\mathcal{P}'$-terminating). Therefore, by construction of $S(L)$, there is no input transition departing from the local state of A in s. □

[5] The theorem in [1] describes a different condition, **CC2'**, which is easier to implement and is equivalent to **CC2** when in conjunction with **CC3**

ACCEPTED MANUSCRIPT

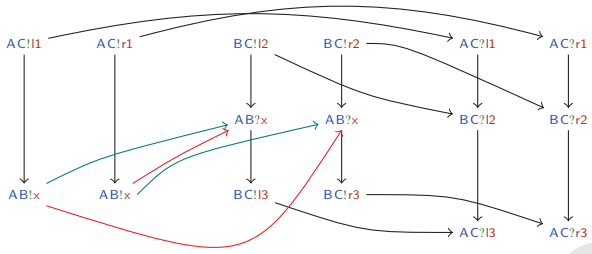


Figure 2: Inter-participant closure of pomset of Eq. (12)

6. Pomset-based verification conditions

We introduce a different approach to check realisability and sound termination of specifications, which does not require to explicitly compute the language of the family of pomsets. This allows us to avoid the combinatorial explosion due to interleavings. The main strategy is to provide alternative definitions of closures directly on pomsets which handle both *intra-* and *inter-participant* concurrency. Besides theoretical benefits, this yields a clear advantage for practitioners. In fact, design errors can be identified and confined in more abstract models, closer to the global specification than to traces of execution. Also, our verification conditions require to analyze sets of pomsets; therefore, they are syntax-oblivious. As discussed in Section 8, our conditions strictly entail the corresponding ones in Section 5

Definition 7 (Closure)**.** *Let* ρ *be a function from* $\mathcal{P}$ *to pomsets and* $(r^{\mathsf{A}})_{\mathsf{A}\in\mathcal{P}}$ *be the tuple where* $r^{\mathsf{A}} = \rho(\mathsf{A})\restriction_{\mathsf{A}}$ *for all* $\mathsf{A}\in\mathcal{P}$*. The inter-participant closure* $\square((r^{\mathsf{A}})_{\mathsf{A}\in\mathcal{P}})$ *is the set of all well-formed pomsets* $[\bigcup_{\mathsf{A}\in\mathcal{P}}\mathcal{E}_{r^{\mathsf{A}}},\ \leq_I \cup \bigcup_{\mathsf{A}\in\mathcal{P}}\leq_{r^{\mathsf{A}}},\ \bigcup_{\mathsf{A}\in\mathcal{P}}\lambda_{r^{\mathsf{A}}}]$ *where* $\leq_I \subseteq \{(e^{\mathsf{A}},e^{\mathsf{B}})\in\mathcal{E}_{r^{\mathsf{A}}}\times\mathcal{E}_{r^{\mathsf{B}}},\mathsf{A},\mathsf{B}\in\mathcal{P} \mid \lambda_{r^{\mathsf{A}}}(e^{\mathsf{A}}) = \mathsf{AB!m}, \lambda_{r^{\mathsf{B}}}(e^{\mathsf{B}}) = \mathsf{AB?m}\}$.

Informally, the inter-participant closure takes one pomset for every participant and generates all "acceptable" matches between output and input events. We use Eq. (12) and Fig. 2 to illustrate the inter-participant closure. The singleton $[\![\mathsf{G}_{(12)}]\!]$ contains one pomset that is the composition of two independent pomsets, which intuitively represent two concurrent "threads". The first thread, dubbed $r_{(12)_a}$, is made of the eight leftmost events in Eq. (12) and the second thread, dubbed $r_{(12)_b}$, is made of the eight rightmost events in Eq. (12). Let r^{R} be the projection of the single pomset in $[\![\mathsf{G}_{(12)}]\!]$ for $\mathsf{R}\in\mathcal{P}$, then the inter-participant closure of $(r^{\mathsf{R}})_{\mathsf{R}\in\mathcal{P}}$ consists of the two well-formed pomsets of Fig. 2, the one that uses the black and green dependencies, and the one that uses the black and red dependencies. Notice that the order $\leq_I$ in Definition 7 is a subset of the product of outputs and matching inputs and this the closure to contain only well-formed pomsets. For example, the closure of $[\![\mathsf{G}_{(12)}]\!]$ does not contain the pomset having both green and red arrows.

Definition 8. *A* pomset r is *less permissive* than pomset r' *(or* r' is more permissive than r*, written* $r\sqsubseteq r'$*) when* $\mathcal{E}_r = \mathcal{E}_{r'}$, $\lambda_r = \lambda_{r'}$, *and* $\leq_r \supseteq \leq_{r'}$.

ACCEPTED MANUSCRIPT

Lemma 1. *If $r \sqsubseteq r'$ then $\mathbb{L}(r) \subseteq \mathbb{L}(r')$.*

Definition 9 (CC2-POM). *A set of pomsets R over $\mathcal{L}$ satisfies closure condition **CC2-POM** if for all tuples $(r^{\mathsf{A}})_{\mathsf{A}\in\mathcal{P}}$ of pomsets of R, for every pomset $r \in \Box((r^{\mathsf{A}}\downarrow_{\mathsf{A}})_{\mathsf{A}\in\mathcal{P}})$, there exists $r' \in R$ such that $r \sqsubseteq r'$.*

Intuitively, Definition 9 requires that if all the possible executions of a pomset cannot be distinguished by any of the participants of R, then those executions must be part of the language of R. Theorem 5 below shows that **CC2-POM** entails **CC2**; its proof is based on "counting" the number of events with a certain label l preceding an event e in the order $\leq_r$ of a pomset r: we write $\mathsf{card}^r_l(e)$ for such number (namely, $\mathsf{card}^r_l(e)$ is the cardinality of $\{e' \in \mathcal{E}_r \mid e' \leq_r e \wedge \lambda_r(e') = l\}$).

Theorem 5. *If R satisfies **CC2-POM** then $\mathbb{L}(R)$ satisfies **CC2**.*

Proof. Let w be a well-formed and complete word over $\mathcal{L}$ that satisfies hypothesis of **CC2**: for every participant $\mathsf{A} \in \mathcal{P}$ there exists $w^{\mathsf{A}} \in \mathbb{L}(R)$ for which $w\downarrow_{\mathsf{A}} = w^{\mathsf{A}}\downarrow_{\mathsf{A}}$. Then, for each $\mathsf{A} \in \mathcal{P}$, there is a pomset $r^{\mathsf{A}} \in R$ such that a linearization ℓ_{A} of r^{A} yields w^{A}. We can hence take the pomset

$$r = \left[\bigcup_{\mathsf{A}\in\mathcal{P}} \mathcal{E}_{r^{\mathsf{A}}\downarrow_{\mathsf{A}}},\quad \leq_I \cup \bigcup_{\mathsf{A}\in\mathcal{P}} \leq_{r^{\mathsf{A}}\downarrow_{\mathsf{A}}},\quad \bigcup_{\mathsf{A}\in\mathcal{P}} \lambda_{r^{\mathsf{A}}\downarrow_{\mathsf{A}}}\right]$$

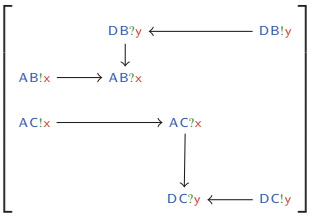
where

$$\leq_I = \bigcup_{\mathsf{B}\neq\mathsf{A}\in\mathcal{P}} \left\{ (e^{\mathsf{A}}, e^{\mathsf{B}}) \in \mathcal{E}_{r^{\mathsf{A}}\downarrow_{\mathsf{A}}} \times \mathcal{E}_{r^{\mathsf{B}}\downarrow_{\mathsf{B}}} \;\middle|\; \begin{array}{l} \lambda_{r^{\mathsf{A}}}(e^{\mathsf{A}}) = \mathsf{AB!m} \text{ and } \lambda_{r^{\mathsf{B}}}(e^{\mathsf{B}}) = \mathsf{AB?m} \\ \text{and } \mathsf{card}^{r^{\mathsf{A}}\downarrow_{\mathsf{A}}}_{\mathsf{AB!m}}(e^{\mathsf{A}}) = \mathsf{card}^{\ell_{r^{\mathsf{B}}}\downarrow_{\mathsf{B}}}_{\mathsf{AB?m}}(e^{\mathsf{B}}) \end{array} \right\}$$

The pomset r is in $\Box((r^{\mathsf{A}}\downarrow_{\mathsf{A}})_{\mathsf{A}\in\mathcal{P}})$, since it is well-formed and complete and $\leq_I$ satisfies conditions of Definition 7. In fact, since w is well-formed and complete, all send and receive events have corresponding matching events. Also by construction, $w \in \mathbb{L}(r)$ and, for every A, $r\downarrow_{\mathsf{A}} \sqsubseteq r^{\mathsf{A}}\downarrow_{\mathsf{A}}$. Finally, by **CC2-POM** there exists $r' \in R$ such that $r \sqsubseteq r'$, therefore $w \in \mathbb{L}(r')$ hence $w \in \mathbb{L}(R)$. □

Theorems 5 and 1 demonstrate that **CC2-POM** is a sufficient condition for weak-ralisability. The families of pomsets $[\![\mathsf{G}_{(4)}]\!]$, $[\![\mathsf{G}_{(6)}]\!]$, $[\![\mathsf{G}_{(7)}]\!]$, $[\![\mathsf{G}_{(8)}]\!]$, $[\![\mathsf{G}_{(9)}]\!]$, $[\![\mathsf{G}_{(10)}]\!]$, $\{r_{13.a}\}$, and R_{13} of Section 4 satisfy **CC2-POM**. For the other families of pomsets, we report one of the pomsets in the inter-participant closure that does meet the closure condition:

- for $[\![\mathsf{G}_{(11)}]\!]$ the following pomset captures the case when B and C do not agree on the order of message reception:



ACCEPTED MANUSCRIPT

- for $[\![\mathsf{G}_{(12)}]\!]$ the pomset of Fig. 2 that uses the black and red dependencies, which represents the case when the right thread "steals" the message x generated by the left thread.

The next condition requires to introduce the concept of *prefix* of a pomset r, which is a pomset r' on a subset of the events of r that preserves the order and labelling of r; formally (following [18])

Definition 10 (Prefix pomsets)**.** *A pomset $r' = [\mathcal{E}', \leq', \lambda']$ is a* prefix *of pomset $r = [\mathcal{E}, \leq, \lambda]$ if there exists a label preserving injection $\phi : \mathcal{E}' \to \mathcal{E}$ such that $\phi(\leq') = \leq \cap (\mathcal{E} \times \phi(\mathcal{E}'))$*

We remark that an arbitrary sub-pomset satisfies the weaker condition $\phi(\leq') = \leq \cap (\phi(\mathcal{E}') \times \phi(\mathcal{E}'))$. Instead, $\phi(\leq') = \leq \cap (\mathcal{E} \times \phi(\mathcal{E}'))$ prevents events in $\mathcal{E} \setminus \phi(\mathcal{E}')$ from preceding events in $\phi(\mathcal{E}')$ and it is equivalent to say that for all $e' \in \mathcal{E}'$ if there is $e \leq \phi(e')$ then there exists $e'' \in \mathcal{E}'$ such that $\phi(e'') = e$ and $e'' \leq' e'$.

Lemma 2. *Let r be a pomset over $\mathcal{L}$ and w be a word in $\mathcal{L}^\star$, $w \in \mathsf{pref}(\mathbb{L}(r))$ if, and only if, there exists a prefix r' of r such that $w \in \mathbb{L}(r')$.*

Definition 11 (**CC3-POM**)**.** *A set of pomsets R over $\mathcal{L}$ satisfies closure condition **CC3-POM** if for all tuples of pomsets $(\bar{r}^{\mathsf{A}})_{\mathsf{A} \in \mathcal{P}}$ such that $\bar{r}^{\mathsf{A}}$ is a prefix of a pomset $r^{\mathsf{A}} \in R$ for every A, and for every pomset $\bar{r} \in \square((\bar{r}^{\mathsf{A}}\!\downarrow_{\mathsf{A}})_{\mathsf{A} \in \mathcal{P}})$ there is a pomset $r' \in R$ and a prefix $\bar{r}'$ of r' such that $\bar{r} \sqsubseteq \bar{r}'$.*

Theorem 6. *If R satisfies **CC3-POM** then $\mathbb{L}(R)$ satisfies **CC3**.*

Proof. Let w be a word that satisfies hypothesis of **CC3**: for every participant $\mathsf{A} \in \mathcal{P}$, there exists a word $w^{\mathsf{A}} \in \mathsf{pref}(\mathbb{L}(R))$ such that $w\!\downarrow_{\mathsf{A}} = w^{\mathsf{A}}\!\downarrow_{\mathsf{A}}$. Therefore, there is a pomset $\bar{r}^{\mathsf{A}}$ prefix of a pomset $r^{\mathsf{A}} \in R$ such that $w^{\mathsf{A}} \in \mathbb{L}(\bar{r}^{\mathsf{A}})$ and let ℓ_{A} be one of the linearizations of $\bar{r}^{\mathsf{A}}$ that corresponds to w^{A}. Define

$$\bar{r} = \left[\bigcup_{\mathsf{A} \in \mathcal{P}} \mathcal{E}_{\bar{r}^{\mathsf{A}}\!\downarrow_{\mathsf{A}}}, \qquad \leq_{\mathit{I}} \cup \bigcup_{\mathsf{A} \in \mathcal{P}} \left(\leq_{\bar{r}^{\mathsf{A}}\!\downarrow_{\mathsf{A}}} \right), \quad \bigcup_{\mathsf{A} \in \mathcal{P}} \lambda_{\bar{r}^{\mathsf{A}}\!\downarrow_{\mathsf{A}}} \right]$$

where

$$\leq_{\mathit{I}} = \bigcup_{\mathsf{B} \neq \mathsf{A} \in \mathcal{P}} \left\{ (e^{\mathsf{A}}, e^{\mathsf{B}}) \in \mathcal{E}_{\bar{r}^{\mathsf{A}}\!\downarrow_{\mathsf{A}}} \times \mathcal{E}_{\bar{r}^{\mathsf{B}}\!\downarrow_{\mathsf{B}}} \;\middle|\; \begin{array}{l} \lambda_{\bar{r}^{\mathsf{A}}}(e^{\mathsf{A}}) = \mathsf{A}\,\mathsf{B}!\mathsf{m} \text{ and } \lambda_{\bar{r}^{\mathsf{B}}}(e^{\mathsf{B}}) = \mathsf{A}\,\mathsf{B}?\mathsf{m} \\ \text{and } \mathsf{card}^{\bar{r}^{\mathsf{A}}\!\downarrow_{\mathsf{A}}}_{\mathsf{A}\,\mathsf{B}!\mathsf{m}}(e^{\mathsf{A}}) = \mathsf{card}^{\bar{r}^{\mathsf{B}}\!\downarrow_{\mathsf{B}}}_{\mathsf{A}\,\mathsf{B}?\mathsf{m}}(e^{\mathsf{B}}) \end{array} \right\}$$

The pomset $\bar{r}$ is in $\square((\bar{r}^{\mathsf{A}}\!\downarrow_{\mathsf{A}})_{\mathsf{A} \in \mathcal{P}})$, since it is well-formed and $\leq_{\mathit{I}}$ satisfies conditions of Definition 7. In fact, since w is well-formed, all receives have matching sends. Also by construction, $w \in \mathbb{L}(\bar{r})$ and, for every A, $\bar{r}\!\downarrow_{\mathsf{A}} \sqsubseteq \bar{r}^{\mathsf{A}}\!\downarrow_{\mathsf{A}}$. Hence, by **CC3-POM** there exists $r' \in R$ and a prefix $\bar{r}'$ of r such that $\bar{r} \sqsubseteq \bar{r}'$, therefore $w \in \mathbb{L}(\bar{r}')$ and therefore $w \in \mathsf{pref}(\mathbb{L}(R))$. □

From Theorems 2, 5 and 6, it follows that if a set of pomsets R satisfies **CC2-POM** and **CC3-POM** then $\mathbb{L}(R)$ is safe realisable (notice that **CC3-POM** alone is not a sufficient condition for safe realisability, [22] demonstrates that both **CC2** and **CC3** are necessary).

ACCEPTED MANUSCRIPT

The families of pomsets $[\![\mathsf{G}_{(4)}]\!]$, $[\![\mathsf{G}_{(6)}]\!]$, $[\![\mathsf{G}_{(7)}]\!]$, $[\![\mathsf{G}_{(8)}]\!]$, and $\{r_{13.a}\}$ of Section 4 satisfy **CC3-POM**. Families of pomsets $[\![\mathsf{G}_{(11)}]\!]$ and $[\![\mathsf{G}_{(12)}]\!]$ are not realisable, since they do not satisfy **CC2-POM**. For the other families of pomsets, we report one of the pomsets in the inter-participant closure of the prefixes that does meet the condition of **CC3-POM**:

- for $[\![\mathsf{G}_{(9)}]\!]$ the pomset $\left[\begin{smallmatrix}\mathsf{AB!x} & \mathsf{CD!x}\end{smallmatrix}\right]$ represents the case when A and D do not agree on which participant should communicate
- for $[\![\mathsf{G}_{(10)}]\!]$ the following pomset represents the case A and C do not agree on the message to deliver

$$\left[\begin{array}{ll}\mathsf{AB!y} \longrightarrow \mathsf{AB?y} & \mathsf{CB!x} \\ \downarrow & \\ \mathsf{AB!z} & \end{array}\right]$$

- for R_{13} the following pomset represents the case A and C do not agree on the message to deliver

$$\left[\begin{smallmatrix}\mathsf{AB!x} \longrightarrow \mathsf{AB?x} & \mathsf{CB!z}\end{smallmatrix}\right]$$

Like for the closure conditions, we lift the sufficient condition for termination soundness to pomsets.

Definition 12 (Terminating pomsets)**.** *A participant* $\mathsf{A} \in \mathcal{P}$ *is* termination-unaware *for a set of pomsets* R *if there are* $r, r' \in R$*, and a label-preserving injection* $\phi : \mathcal{E}_{r\downarrow_\mathsf{A}} \to \mathcal{E}_{r'\downarrow_\mathsf{A}}$ *such that* $\leq = \phi(\leq_{r\downarrow_\mathsf{A}}) \cup \leq_{r'\downarrow_\mathsf{A}}$ *is a partial order and*

$$min_{\leq}(\mathcal{E}_{r'\downarrow_\mathsf{A}}) \subseteq \phi(min_{\leq_{r\downarrow_\mathsf{A}}}(\mathcal{E}_{r\downarrow_\mathsf{A}})) \qquad \text{and} \qquad min_{\leq}(\mathcal{E}_{r'\downarrow_\mathsf{A}} \setminus \phi(\mathcal{E}_{r\downarrow_\mathsf{A}})) \cap \mathcal{L}^{?} \neq \emptyset$$

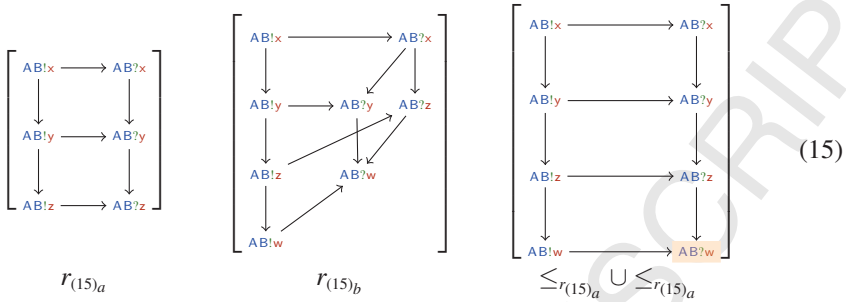
Given a set of participants $\mathcal{P}' \subseteq \mathcal{P}$*, we say that* R *is* $\mathcal{P}'$-terminating *when there is no* $\mathsf{A} \in \mathcal{P}'$ *termination-unaware for* R*.*

Theorem 7. *Given* $\mathcal{P}' \subseteq \mathcal{P}$*, if* R *is* $\mathcal{P}'$-terminating *then* $\mathbb{L}(R)$ *is* $\mathcal{P}'$-terminating.

Proof. Given a word $w \in \mathbb{L}(R)$, there is a pomset $r \in R$ such that $w \in \mathbb{L}(R)$. Let $\mathsf{A} \in \mathcal{P}'$ and assume that there is $w' \in \mathbb{L}(R)$ such that $w\downarrow_\mathsf{A}$ is a prefix of $w'\downarrow_\mathsf{A}$. Therefore, there is a pomset $r' \in R$ such that $w'\downarrow_\mathsf{A} \in \mathbb{L}(r'\downarrow_\mathsf{A})$. Let $e_1, \ldots, e_n$ and $e'_1, \ldots, e'_{n'}$, with $n < n'$, be the linearizations of $\leq_r$ and $\leq_{r'}$ respectively for the world w and w' respectively. Let ϕ be the injection that maps e_i to e'_i for $1 \leq i \leq n$, then $\leq = \phi(\leq_{r\downarrow_\mathsf{A}}) \cup \leq_{r'\downarrow_\mathsf{A}}$ is a partial order. Therefore $min_{\leq}(\mathcal{E}_{r'\downarrow_\mathsf{A}} \setminus \phi(\mathcal{E}_{r\downarrow_\mathsf{A}})) \cap \mathcal{L}^{?} \neq \emptyset$ since R is $\mathcal{P}'$*-terminating*, thus the first symbol of w' after w cannot be an input. □

We use Eq. (15) to describe termination awareness. B is *termination-unaware* for the set of pomsets $[\![\mathsf{G}_{(15)}]\!]$. In fact, let $\phi : \mathcal{E}_{r_{(15)_a}\downarrow_\mathsf{B}} \to \mathcal{E}_{r_{(15)_b}\downarrow_\mathsf{B}}$ be the only possible label-preserving injection, then $\leq = \phi(\leq_{r_{(15)_a}\downarrow_\mathsf{B}}) \cup \leq_{r_{(15)_b}\downarrow_\mathsf{B}}$ is the partial order in Eq. (15).c, and $min_{\leq}(\mathcal{E}_{r_{(15)_b}\downarrow_\mathsf{B}} \setminus \phi(\mathcal{E}_{r_{(15)_a}\downarrow_\mathsf{B}})) = \{\mathsf{AB?w}\}$ is not disjoint from $\mathcal{L}^{?}$. Intuitively, $\leq$ represents the intersection of the languages of the two pomsets $r_{(15)_b}\downarrow_\mathsf{B}$ and $r_{(15)_a}\downarrow_\mathsf{B}$.

ACCEPTED MANUSCRIPT



(15)

The families of pomsets $[\![\mathsf{G}_{(4)}]\!]$, $[\![\mathsf{G}_{(6)}]\!]$, $[\![\mathsf{G}_{(10)}]\!]$, $[\![\mathsf{G}_{(11)}]\!]$, and $[\![\mathsf{G}_{(12)}]\!]$ are *terminating* for all principals.

- $[\![\mathsf{G}_{(7)}]\!]$ is not terminating for B, since the projection of the right pomset on B is the empty pomset, this is a prefix of the projection of the left pomset on B, and the first non-common event for B is the input A B?x
- $[\![\mathsf{G}_{(8)}]\!]$ is not terminating for B (and C), since the projection of the left pomset on B is a prefix of the projection of the right pomset on B, and the first non-common event for B is the input A B?y (the input B C?z for C)

7. Verifying our closure conditions

Checking **CC2-POM** and **CC3-POM** is decidable since we assume R to be a finite set of finite pomsets and $\mathcal{P}$ to be finite. For **CC2-POM**, there are finite tuples $(r^{\mathsf{A}})_{\mathsf{A}\in\mathcal{P}}$ of pomsets of R and for each tuple the inter-participant closure is a finite set of finite pomsets. For **CC3-POM**, the number of prefixes of pomsets in R is also finite. Sections 7.1 and 7.2 describe the algorithms[6]. Section 7.3 benchmarks our prototype on some of the examples in Section 4, while Section 7.4 evaluates how it performs on a simple yet realistic scenario.

7.1. Auxiliary operations

The key algorithm in the pseudo-code in Fig. 3 is the one at lines 9-19. Basically, `inter_participant_closure` computes the inter-participant closure assuming that the input tuple $t = (r_{\mathsf{A}})_{\mathsf{A}\in\mathcal{P}}$ consists of pomsets such that A is the subject of all the events in r_{A}. Before commenting on `inter_participant_closure`, we focus on the other two algorithms in Fig. 3.

Given a (DAG representing a) pomset and two disjoint lists of nodes, the function `connect_linearizations` (lines 1-2 in Fig. 3) extends the order of the pomset adding the relations between the i-th event of $\vec{e}_1$ and $\vec{e}_2$. This function is used to add the dependencies constraining the outputs $\vec{e}_1$ to precede the corresponding inputs in $\vec{e}_2$.

[6]Our prototype implementation is available at `https://bitbucket.org/guancio/chosem-tools`.

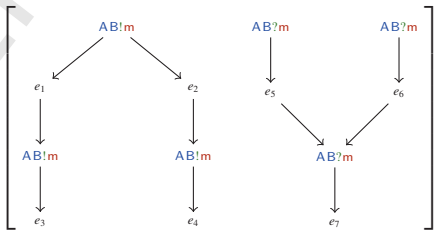
ACCEPTED MANUSCRIPT

```
1  connect_linearizations([E, ≤, λ], ē1, ē2)
2    return (E, (≤ ∪_{i∈0..min(|ē1|,|ē2|)−1} (ē1[i], ē2[i]))*, λ)
3
4  all_graphs_for_input(r, AB?m):
5    Ls = lin(r↓_{AB!m})
6    Lr = lin(r↓_{AB?m})
7    return ⋃_{ē1,ē2∈Ls×Lr} connect_linearizations(r, ē1, ē2)
8
9  inter_participant_closure(t):
10   r = ∪_{A∈P} t(A)
11   if ∃l ∈ L?.#(r, AB?m) > #(r, AB!m)
12     return ∅
13   R = {r}
14   for AB?m ∈ L?:
15     R' = ∅
16     for r ∈ R:
17       R' += all_graphs_for_input(r, AB?m)
18     R = R'
19   return R
```

Figure 3: Implementation of inter-participant closure for one tuple of pomsets

Note that the result is a pomset when $\vec{e}_1 \cup \vec{e}_2$ is a list of events of $\mathcal{E}$ both not containing duplicated events and such that $\vec{e}_2[i] \not\leq \vec{e}_1[i]$ for every $0 \leq i < min(|\vec{e}_1|, |\vec{e}_2|)$.

Given a pomset r, the function `all_graphs_for_input` (lines 4-7 in Fig. 3) computes all possible ways the events labeled with output A B!m can be connected to a corresponding input event while preserving the dependencies of participant A and B. In fact, when computing the inter-participant closure, if the same message is exchanged multiple times then there can be multiple ways to obtain a well-formed pomset. For example, in the pomset



there are four possible ways to order events labelled with A B!m with those labelled with A B?m. The function computes all possible linearizations of the outputs, all linearizations of the inputs (which therefore respect the dependencies of the sender and receiver), and generates a new pomset for each pair of linearizations by using `connect_linearizations`.

We return now to the key algorithm of Fig. 3. The inter-participant closure is obtained by iterating the `all_graphs_for_input` procedure on each input label. Notice that the

ACCEPTED MANUSCRIPT

function `inter_participant_closure` returns an empty set of pomsets if there exists an input label that occurs more often than the corresponding output label. This and the use of linearizations ensure that the resulting set contains only well-formed pomsets.

Computing the inter-participant closure is in general expensive due to the combinatorial explosion of the possible linearizations of matching events. This is due to the presence of multiple and independent instances of the same action.

Definition 13. *Let r be a pomset over $\mathcal{L}$. Two events $e_1, e_2 \in \mathcal{E}_r$ are* concurrent repetitions *if*

- $\lambda_r(e_1) = \lambda_r(e_2)$
- *neither* $e_1 \leq_r e_2$ *nor* $e_2 \leq_r e_1$
- *for all predecessors* $e \leq_r e_1$, *events* e, e_2 *are not concurrent repetitions*
- *for all predecessors* $e \leq_r e_2$, *events* e_1, e *are not concurrent repetitions*

We use $\sim$ to identify the largest equivalence relation that ensures if $e_1 \sim e_2$ than either $e_1 = e_2$ or e_1 and e_2 are concurrent repetitions. An action $l \in \mathcal{L}$ concurrently repeats in *r if there exist $e \neq e'$ such that $\lambda_r(e) = \lambda_r(e') = l$ and $e_1 \sim e_2$.*

Since r is acyclic, then the equivalence classes of $\mathcal{E}_{r/\sim}$ that have the same label are totally ordered. For a tuple t, let $r = \cup_{A \in \mathcal{P}} t(A)$ be the union of the participant's branches, the number of pomsets in the inter-participant closure is proportional to $\prod_{\mathcal{E} \in \mathcal{E}_{r/\sim}} 2^{\mathrm{card}(\mathcal{E})}$. Therefore, if there are few concurrently repeating actions then the size of the inter-participant closure is small (e.g., there is at most one pomset in the closure of a pomset with no concurrently repeating actions).

In practice, the presence of actions that concurrently repeat is limited. The scenario in Section 7.4 illustrates this point. Some specification formalisms even impose conditions that syntactically avoid this issue (e.g. see well-forkedness of [31] or the even more restrictive conditions of e.g., [17]). In fact, sending the same message in two independent threads may "confuse" receivers making it hard (or impossible) to decide which receiving thread should consume the message and possibly leading to coordination problems.

7.2. Checking pomset-based closures

The pseudo-code in Fig. 4 checks condition **CC2-POM** for a set of pomsets R. To do this, `CC2-POM` uses the function

```
1 get_all_branches(R):
2   for A ∈ P:
3     branches[A] = ∅
4     for r ∈ R:
5       if not exist_more_permissive(r↓A, branches[A]):
6         branches[A] += r↓A
7   return branches
```

that retrieves all branches of every participant. This function filters out the branches of participants by using

ACCEPTED MANUSCRIPT

```
1 CC2-POM(R):
2   branches = get_all_branches(R)
3   tuples = branches[A_1] × ... × branches[A_n]
4   ipc = ∪_{t∈tuples} inter_process_closure(t)
5   for r ∈ ipc:
6     if not exist_more_permissive(r, R):
7       return false
8   return true
```

Figure 4: Verification of **CC2-POM**

```
1 exist_more_permissive(r, R):
2   return ∃r' ∈ R | subgraph_is_isomorphic(r', r)
```

that avoids adding duplicate pomsets and to reduce the number of pomsets used for the inter-participant closure. For example, participants A and D have the same behavior in both the pomsets of $[\![G_{(11)}]\!]$.

The cartesian product (line 3 of Fig. 4) of the principal branches returns a set whose each element is a tuple that has one pomset per principal.

Similarly to the case of inter-participant closure, the cost of checking **CC2** depends on the presence of concurrently repeated actions. The complexity of finding a label-preserving graph isomorphism (`subgraph_is_isomorphic`) is exponential in the number of events. More precisely, let $e_1 \approx e_2$ be the equivalence relation $\lambda_r(e_1) = \lambda_r(e_2)$, and let $\mathcal{E}_{r/\approx}$ be the corresponding equivalence classes. The complexity of finding a label-preserving isomorphism is $\prod_{E \in \mathcal{E}_{r\approx}} 2^{\mathrm{card}(E)}$. However, since the graphs are acyclic, the complexity can be bound to the number of concurrently-repeated actions. In fact, the classes of $\mathcal{E}_{r/\sim}$ that have the same label are totally ordered. Therefore for every label l, a graph isomorphism between r and r' can map events in the i-th equivalence class of $\mathcal{E}_{r/\sim}$ having label l only to events in the i-th equivalence class of $\mathcal{E}_{r'/\sim}$ having label l. For this reason the complexity of finding a label-preserving isomorphism is exponential in $\prod_{\mathcal{E} \in \mathcal{E}_{r/\sim}} 2^{\mathrm{card}(\mathcal{E})}$. For this reason, if there are no concurrently repeated actions in R then checking **CC2-POM** can be done in polynomial time with respect to the number of events.

Finally, algorithm in Fig. 5 checks condition **CC3-POM** for a set of pomsets R. The function `get_all_prefixes` returns all prefixes of a pomset by simply iterating all possible subsets of events that satisfies the dependencies.

7.3. Benchmarking our algorithms

To assess how our algorithms perform we implemented a prototype tool. Our tool is written in Python and relies on the NetworkX package for graph operations. In fact, pomsets are represented as direct labelled acyclic graphs to implement the algorithms presented in Sections 7.1 and 7.2. The main NetworkX's primitive used in the prototype is `subgraph_is_ismorphic`(r_1,r_2), which returns true iff r_1 and r_2 have the same

ACCEPTED MANUSCRIPT

```
1  get_all_prefixes(E, ≤, λ):
2    E' = E
3    to_process = {∅}
4    prefixes = ∅
5    while (to_process ≠ ∅):
6        prefix = to_process.pop()
7        for e ∈ E'\prefix:
8            if {e' | e' ≤ e}\prefix = ∅:
9              to_process += prefix∪{e}
10       prefixes += [prefix, ≤, λ]
11   return prefixes
12
13
14 CC3-POM(R):
15   branches = get_all_branches(R)
16   for A ∈ P:
17     prefixes[A] = ∪_{r∈branches[A]} get_all_prefixes(r)
18   tuples = prefixes[A_1] × ... × prefixes[A_n]
19   ipc = ∪_{t∈tuples} inter_process_closure(t)
20   R' = ∪_{r∈R} get_all_prefixes(r)
21   for r ∈ ipc:
22     if not exist_more_permissive(r, R'):
23       return false
24   return true
```

Figure 5: Verification of **CC3-POM**

ACCEPTED MANUSCRIPT

	CC2-POM					CC3-POM					CC3 (DS)	
	B	N	I	E	T	B	N	I	E	T	M	T
$[\![G_{(4)}]\!]$	4	4	2	0	1	6	9	5	0	2	2	0
$[\![G_{(6)}]\!]$	6	8	2	0	3	11	48	9	0	11	2	0
$[\![G_{(7)}]\!]$	6	8	2	0	1	7	12	5	0	2	0	0
$[\![G_{(8)}]\!]$	6	8	2	0	2	14	64	27	0	46	12	1
$[\![G_{(9)}]\!]$	8	16	4	1	3	8	16	9	4	5	0	0
$[\![G_{(10)}]\!]$	6	8	2	0	3	16	120	38	10	64	3	0
$[\![G_{(11)}]\!]$	6	4	4	2	9	18	400	100	18	340	8	1
$[\![G_{(12)}]\!]$	3	1	4	2	16	0	2304	668	258	9297	2400	13978

Table 1: Benchmarks: B: average distinct branches per principal; N: number of tuples; I: number of pomets in the inter-participant closure; E: number of pomsets that do not satisfy the condition; T: total time in milliseconds; M: number of Equivalent MSCs; T **CC3**: milliseconds to initialize the algorithm to check **CC3** on MS Cs

number of nodes and r_1 is isomorphic to a subgraph of r_2. If two graphs represent pomsets then the predicates holds iff $r_2 \sqsubseteq r_1$.

One of the advantages of checking **CC⋆-POM** instead of **CC⋆** is that the former does not require the explicit computation of the language of the family of pomsets, which can lead to combinatorial explosion due to interleavings. For example, $[\![G_{(12)}]\!]$ contains one pomset and has two actions that occur concurrently: AB!x and AB?x . Therefore the inter-participant closure has two pomsets (see Fig. 2). Checking the relation $\_ \sqsubseteq \_$ between these pomsets and the pomset in $[\![G_{(12)}]\!]$, requires to iterate over all possible label preserving isomorphisms. However, since all actions except AB!x and AB?x do not occur concurrently, there are only two of such isomorphisms. Checking **CC⋆** can be more expensive. The left and right subpomsets of Eq. (12), which represent the two threads, have 32 different linearizations, each one consisting of 8 events. Therefore the language of $[\![G_{(12)}]\!]$ consists of $32 * 32 * 2^8 = 2^{18}$ words. For this reason, directly analyzing the inter-participant closure in Fig. 2 is more efficient than generating the languages.

We collected some measurements on the performance of our prototype in Table 1. The experiments have been executed on a Intel 2.2 Ghz i7 with 16 GB of RAM. The table reports the outcome for the evaluation of **CC⋆-POM** for some of the examples presented earlier. The prototype identifies several errors for scenarios that are not safely realisable, reporting the pomsets that do not meet the closure conditions as counterexamples. As shown in Table 1, the evaluation of closure conditions is pretty fast for simple examples. However, the number of prefixes to check in **CC3-POM** can be large when participants have several concurrent threads. It is worth noticing that for $[\![G_{(11)}]\!]$ and $[\![G_{(12)}]\!]$ the realisability check can terminate before checking **CC3-POM** since these examples do not satisfy **CC2-POM**. The last two columns of the table report metrics to compare our approach with respect to checking **CC3** using an implementation that we devised following the algorithm described in [1]. For this algorithm, the set of pomsets must be expressed via MSCs, whose participant's projections are totally ordered pomsets. For this reason, the number of MSCs that must be analyzed (column "M") is large when the

ACCEPTED MANUSCRIPT

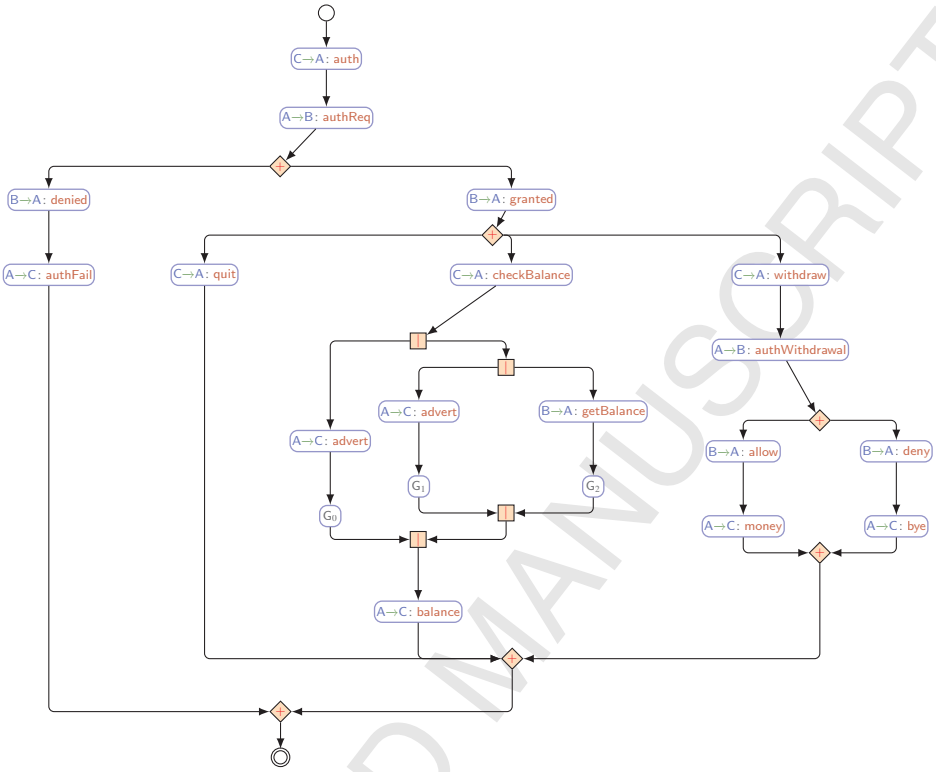


Figure 6: The ATM choreography

original pomsets have intra-participant concurrency. We implemented only the part of the algorithm in [1] for the generation of the data structures (DS) necessary to check **CC3**, for this reason the time reported in the last column represents only the time needed to generate the MSCs and to prepare the indexing tables used by the algorithm, and it does not account for the actual time needed to check the closure condition. As shown in Table 1, the time to generate the DS for $[\![G_{(12)}]\!]$ (the last row of Table 1) is higher than the time our prototype takes to check **CC3-POM** (by a factor of 1.5).

7.4. Applying our prototype

In this section we test how our prototype implementation behaves on a simple, yet realistic, scenario where indeed there is a small number of concurrently repeating actions. We consider an application involving three components, a user C (after Carol), an ATM A, and a bank B. The application can be described by the global graph in Fig. 6. Notice that two advertises are delivered concurrently from the ATM to Carol, therefore the actions A C!advert and A C?advert are concurrently repeating. Moreover, we use G_0, G_1, and G_2 as placeholders for different sub-choreographies used to benchmark our implementation.

ACCEPTED MANUSCRIPT

	CC2-POM					CC3-POM					CC3	
$\|G_i\|$	B	N	I	E	T	B	N	I	E	T	M	T
0	5	125	8	0	89	13	1760	102	16	2026	16	4
1	5	125	8	0	109	13	1760	102	16	2262	16	7
2	5	125	8	0	135	23	7296	436	26	19770	2704	21532

Table 2: Benchmarks: $|G_i|$: number of interactions in G_0, G_1, and G_2; B: average distinct branches per principal; N: number of tuples; I: number of pomets in the inter-participant closure; E: number of pomsets that do not satisfy the condition; T: total time in milliseconds; M: number of Equivalent MSCs; T **CC3**: milliseconds to initialize the algorithm to check **CC3** on MSCs

The workflow is pretty standard but for the two (intentional) design glitches described below. Initially, Carol provides her card and credentials to the ATM (message auth) which requires the authentication to the bank (message authReq). The back decides whether to deny or authorise[7] the use of the card; in the former case A informs C that the authorisation has been denied and the process terminates. The branch where the access is granted (message granted) yields the first glitch due to the fact that B's decision is not properly propagated to Carol, which could send the message checkBalance even if the access is denied. This prevents the example to satisfy **CC3-POM**. Basically, the architect forgot to specify that, after the granted message is received from the bank, the options for Carol should be displayed. The second glitch is due to the fact that the bank is not informed when Carol opts to quit the session with the ATM. This can lead the bank to wait indefinitely for a message when the authorization is granted and Carol quits. Therefore this example is not $\{B\}$-*terminating*.

Table 2 reports the benchmarks for checking **CC2-POM** and **CC3-POM** for this example using our prototype Python tool and the time needed to initialize the algorithm to check **CC3** from [1]. The workflow is parameterised with respect to the three sub-choreographies G_0, G_1, and G_2 in order to vary the number of prefixes the algorithm has to check. More precisely, the sub-choreographies have been instantiated with sequential composition of zero, one, and two interactions, where each interaction uses a unique message.

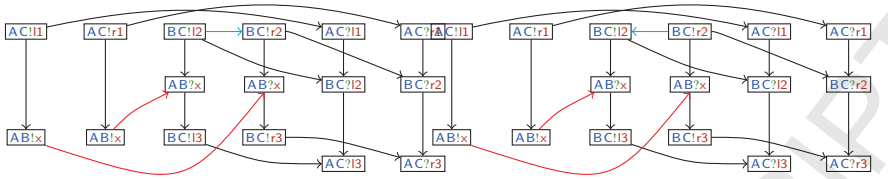
8. Discussion on the pomset based conditions

Section 7.3 shown that checking the verification conditions using pomsets can be more efficient than explicitely generating the languages. Moreover, if a pomset is thought of as the specification of a possible scenario of a system, a further practical advantage of using the conditions of Section 6 is that problems can be discovered at design-time. This permits to easily isolate the problematic scenarios of a specification even if they share multiple traces with non-problematic scenarios.

The fact that **CC⋆-POM** are syntax oblivious allows us to analyse realisability of scenarios independently of the expressiveness of the choreography language. For in-

[7] We assume that all the choices are non-deterministic in order to abstract away from the actual conditions used locally by participants.

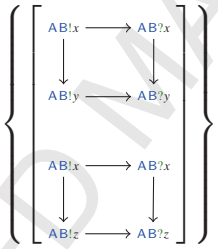
ACCEPTED MANUSCRIPT



(a) $r_{(7)_a}$ (b) $r_{(7)_b}$

Figure 7: A set of pomsets language-equivalent to the pomset with red and black dependencies of Fig. 2, but explicitly interleaves the events BC!l2 and BC!r2 (cyan dependencies)

stance we can conclude that the scenario $\{r_2\}$ is realisable even if it cannot be expressed using the choreographic model of Section 3.2. Also, our conditions allow to establish realisability of the following singleton set of pomsets, which is usually not accepted by specification formalisms because their syntactic driven constrains prevent sending the same message in two independent threads.



As discussed in Section 8, our conditions strictly entail the corresponding ones in Section 5 pomset with green dependencies of Fig. 2. Then, $R_{(2)}$ satisfies **CC2-POM**, since it contains all pomsets that satisfy hypothesis of the closure condition, therefore by Theorem 5 its language satisfies **CC2**. Consider the set $R_{(7)} = \{r_{(7)_a}, r_{(7)_b}, r_{(2)_{green}}\}$, where $r_{(7)_a}$ and $r_{(7)_b}$ are the two pomsets of Fig. 7. Notice that $r_{(7)_a}$ and $r_{(7)_b}$ are equivalent to $r_{(2)_{red}}$, with the exception of the dependency between BC!l2 and BC!r2. Since $r_{(7)_a}$ and $r_{(7)_b}$ have opposite orders between these two events, the union of their languages is equal to the language of $r_{(2)_{red}}$. Therefore the language of $R_{(7)}$ is equal to the language of $R_{(2)}$, hence it also satisfies **CC2**. However, $R_{(7)}$ does not satisfy **CC2-POM**. In fact, the pomset $r_{(2)_{red}}$ satisfies hypothesis of **CC2-POM**, but there is not pomset in $R_{(7)}$ that is more permissive than $r_{(2)_{red}}$.

9. Related work

The surge of *message-passing* applications in industry is revamping the interest for software engineering methodologies supporting designers and developers called to realise communication-centred software. In this context, realisability of global specifications is of concern for both practical and theoretical reasons. Our approach can support choreography languages (e.g. the global graphs used in [31] that allow multi-threaded participants and complex distributed choices). These specifications yield at the

ACCEPTED MANUSCRIPT

same time (*i*) concrete support to scenario-based development, (*ii*) rigorous semantics in terms of partial order of communication events that enable the use of algorithms and tools to reason about and verify communicating applications, and (*iii*) a simple graphical syntax that supports the intuition and makes it easy to practitioners to master the specification without needing to delve into the underlying theory.

A paradigmatic class of such formalisms are *message-sequence charts* (MSCs) [25, 11, 26, 16, 14, 15, 2]. A mechanism to statically detect realisability in MSCs is proposed in [3]. The notions of non-local choices and of termination considered in [3] are less than our verification conditions since intra-participant concurrency is not allowed and termination awareness (Definition 6) is not enforced. In the context of choreographies, several works (e.g., [4, 7, 17]) defined constraints to guarantee the soundness of the projections of global specifications. These approaches address the problem for specific languages, thus these conditions often use information on the syntactical structure of the specification. Instead, conditions presented in Section 6 are syntax-oblivious and they make minimal assumptions on the communication model. Therefore, our results can be applied to a wide range of languages.

The closure conditions reviewed in Section 5 have been initially introduced in [1] to study realisability of MSC. The replacement in the framework of MSC with pomsets is technically straightforward and yields more general results, since it enables multi-threaded participants. In Section 5, to avoid systems where participants can get stuck due to the termination of some partners, we introduce the notion of termination soundness and demonstrate sufficient conditions that guarantee it. Then, we introduce new verification conditions for the distributed realisability of pomsets, which can tame the combinatorial explosion due to the interleaving of communication events.

A problem related to realisability is satisfiability of logical formulae. Model checkers use temporal logic, i.e. LTL, to formalize system specifications. A general problem that must be faced is that formal specifications can be wrong as their implementations. For instance, if a formula is unsatisfiable, then the specification is probably incorrect. Similarly to realisability, the problem of satisfiability of a temporal formula [29] allows to demonstrate that there exists an implementation that meets the specification.

10. Concluding remarks

There are some open questions to address. Pomset semantics of recursive processes is infinite, which precludes to directly use these results for global specifications that have loops. In [5] pomsets were used in combination with proved transition systems to give an non-interleaving semantics of CCS; basically, given a sequence of transitions $p \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} q$ between two CCS processes p and q, a pomset r can be derived from a *proved transition system* so that r represents the equivalence class of traces between p and q "compatible" with traces labelled $\alpha_1, \ldots, \alpha_n$. This work can help us to generalise our results to infinite computations.

Realisability of high-level MSCs has been addressed in [22], but the verification conditions are not syntax-oblivious. The conditions of Section 6 are sufficient but not necessary conditions for realisability. This is due to the fact that the same semantics (i.e., set of traces) can be expressed using different sets of pomsets by exploring different

ACCEPTED MANUSCRIPT

interleavings. We do not know if a notion of normal forms for families of pomsets can be used to guarantee that our conditions are necessary. We conjecture that our semantics could be applied to other coordination paradigms such as order-preserving asynchronous message-passing (as the original semantics of CFSMs), synchronous communications, or tuple based coordination. We leave the exploration of the robustness of our framework as future work.

Our experiments are rather preliminary, but show that our approach outperforms the algorithm in [1]. We plan to complete the implementation of the algorithm in [1] and make a more thorough comparison with our algorithm. The optimisation of our prototype is left as future work. We plan to integrate our prototype into ChorGram [20], a tool we are currently developing, to implement our theoretical framework and apply it to the analysis of global specifications. As discussed in Section 7.4, the execution time grows exponentially in presence of concurrently repeated actions. However, we remark that (1) many choreographic framework restrict the parallel composition of choreographies so that concurrently repeated actions are not allowed and (2) the analysis is done on compact models that may correspond to large pieces of code. For instance, the ATM choreography in Section 7.4 can be compiled in Erlang[8] and the generated code is a few hundred lines long.

[8]ChorGram can generate Erlang executable code from global graphs.

ACCEPTED MANUSCRIPT

References

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

[2] Rajeev Alur, Gerard J. Holzmann, and Doron Peled. An analyzer for message sequence charts. In Tiziana Margaria and Bernhard Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, pages 35–48. Springer, 1996.

[3] Hanêne Ben-Abdallah and Stefan Leue. Syntactic detection of process divergence and non-local choice in message sequence charts. In Ed Brinksma, editor, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 259–274. Springer, 1997.

[4] Laura Bocchi, Hernán C. Melgratti, and Emilio Tuosto. Resolving non-determinism in choreographies. In Zhong Shao, editor, *European Symposium on Programming*, Lecture Notes in Computer Science, pages 493–512. Springer, 2014.

[5] Gérard Boudol and Ilaria Castellani. Permutation of transitions: an event structure semantics for CCS and SCCS. In J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, volume 354 of *Lecture Notes in Computer Science*, pages 411–427. Springer, 1988.

[6] Daniel Brand and Pitro Zafiropulo. On Communicating Finite-State Machines. *Journal of the ACM*, 30(2):323–342, 1983.

[7] Marco Carbone, Kohei Honda, and Nobuko Yoshida. A Calculus of Global Interaction based on Session Types. *Electronic Notes in Theoretical Computer Science*, 171(3):127 – 151, 2007.

[8] Gérard Cécé and Alain Finkel. Verification of programs with half-duplex communication. *I&C*, 202(2):166–190, 2005.

[9] Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty Session Types Meet Communicating Automata. In Helmut Seidl, editor, *European Symposium on Programming*, Lecture Notes in Computer Science, pages 194–213. Springer, 2012.

[10] Haim Gaifman and Vaughan R Pratt. Partial order models of concurrency and the computation of functions. In *Symposium on Logic in Computer Science*, pages 72–85, 1987.

[11] Emmanuel Gaudin and Eric Brunel. Property Verification with MSC. In Ferhat Khendek, Maria Toeroe, Abdelouahed Gherbi, and Rick Reed, editors, *SDL 2013*, Lecture Notes in Computer Science. Springer, 2013.

ACCEPTED MANUSCRIPT

[12] Roberto Guanciale and Emilio Tuosto. An abstract semantics of the global view of choreographies. In Massimo Bartoletti, Ludovic Henrio, Sophia Knight, and Hugo Torres Vieira, editors, *Proceedings 9th Interaction and Concurrency Experience, ICE 2016, Heraklion, Greece, 8-9 June 2016.*, pages 67–82, 2016.

[13] Roberto Guanciale and Emilio Tuosto. Realisability of pomsets via communicating automata. *CoRR*, abs/1810.02469, 2018.

[14] Elsa L. Gunter, Anca Muscholl, and Doron A. Peled. Compositional Message Sequence Charts. In *Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, pages 496–511. Springer, 2001.

[15] Elsa L. Gunter, Anca Muscholl, and Doron A. Peled. Compositional message sequence charts. *International Journal on Software Tools for Technology Transfer*, 5(1):78–89, Nov 2003.

[16] David Harel and Rami Marelly. *Come, let's play: scenario-based programming using LSCs and the play-engine*. Springer, 2003.

[17] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. *Journal of the ACM*, 63(1):9:1–9:67, 2016. Extended version of a paper presented at POPL08.

[18] Joost-Pieter Katoen and Lennard Lambert. Pomsets for message sequence charts. *Formale Beschreibungstechniken für Verteilte Systeme*, pages 197–208, 1998.

[19] Susheel Kumar. 7 Reasons Why Organizations Struggle with Microservices Adoption. `https://blogs.perficient.com/integrate/2017/06/26/7-reasons-why-organization-struggle-with-microservices-adoption/`, 2017.

[20] Julien Lange and Emilio Tuosto. ChorGram. `https://bitbucket.org/emlio_tuosto/chorgram/wiki/Home`.

[21] James Lewis and Martin Fowler. Microservices: a definition of this new architectural term. `http://martinfowler.com/articles/microservices.html`, 2014.

[22] Markus Lohrey. Safe Realizability of High-Level Message Sequence Charts. In Luboš Brim, Mojmír Křetínský, Antonín Kučera, and Petr Jančar, editors, *CONCUR*, Lecture Notes in Computer Science, pages 177–192. Springer, 2002.

[23] Qusay Mahmoud. *Middleware for Communications*. John Wiley & Sons, 2005.

[24] Robin Milner. *Communicating and mobile systems - the Pi-calculus*. Cambridge University Press, 1999.

[25] 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`.

ACCEPTED MANUSCRIPT

[26] Anca Muscholl and Doron Peled. Deciding Properties of Message Sequence Charts. In Stefan Leue and Tarja Johanna Systä, editors, *Scenarios: Models, Transformations and Tools*, pages 43–65. Springer, 2005.

[27] Object Management Group. Business Process Model and Notation. `http://www.bpmn.org`.

[28] Vaughan Pratt. Modeling concurrency with partial orders. *International Journal of Parallel Programming*, 15(1):33–71, 1986.

[29] Kristin Y Rozier and Moshe Y Vardi. LTL satisfiability checking. In Dragan Bošnački and Stefan Edelkamp, editors, *International SPIN Workshop on Model Checking of Software*, Lecture Notes in Computer Science, pages 149–167. Springer, 2007.

[30] Davide Sangiorgi and David Walker. *The π-Calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.

[31] Emilio Tuosto and Roberto Guanciale. Semantics of global view of choreographies. *Journal of Logic and Algebraic Methods in Programming*, 95:17 – 40, 2018.

[32] Web services choreography description language version 1.0. `https://www.w3.org/TR/ws-cdl-10/`, 2005.

ACCEPTED MANUSCRIPT