

Branching pomsets: design, expressiveness and applications to choreographies

Luc Edixhoven^(✉)^{a,b}, Sung-Shik Jongmans^{a,b}, José Proença^c, Ilaria Castellani^d

^a*Department of Computer Science, Open University of the Netherlands, Postbus 2960, 6401 DL, Heerlen, The Netherlands*

^b*Centrum Wiskunde & Informatica (CWI), Postbus 94079, 1090 GB, Amsterdam, The Netherlands*

^c*CISTER, ISEP, Polytechnic Institute of Porto, Rua Dr. António Bernardino de Almeida 431, 4249-015, Porto, Portugal*

^d*INRIA, Université Côte d’Azur, 2004 Route des Lucioles, BP 93, 06902, Sophia Antipolis CEDEX, France*

Abstract

Choreographic languages describe possible sequences of interactions among a set of agents. Typical models are based on languages or automata over sending and receiving actions. Pomsets provide a more compact alternative by using a partial order to explicitly represent causality and concurrency between these actions. However, pomsets offer no representation of choices, thus a set of pomsets is required to represent branching behaviour. For example, if an agent Alice can send one of two possible messages to Bob three times, one would need a set of $2 \times 2 \times 2$ distinct pomsets to represent all possible branches of Alice’s behaviour. This paper proposes an extension of pomsets, named *branching pomsets*, with a branching structure that can represent Alice’s behaviour using $2 + 2 + 2$ ordered actions. We compare the expressiveness of branching pomsets with that of several forms of event structures from the literature. We encode choreographies as branching pomsets and show that the pomset semantics of the encoded choreographies are bisimilar to their operational semantics. Furthermore, we define well-formedness conditions on branching pomsets, inspired by multiparty session types, and we prove that the well-formedness of a branching pomset is a sufficient condi-

Email addresses: led@ou.nl (Luc Edixhoven^(✉)), ssj@ou.nl (Sung-Shik Jongmans), pro@isep.ipp.pt (José Proença), ilaria.castellani@inria.fr (Ilaria Castellani)

tion for the realisability of the represented communication protocol. Finally, we present a prototype tool that implements our theory of branching pomsets, focusing on its applications to choreographies.

Keywords: Choreographies, Pomsets, Realisability, Event structures

1. Introduction

Distributed systems are becoming ever more important. However, designing and implementing them is difficult. The complexity resulting from concurrency and dependencies among agents makes the process error-prone and debugging non-trivial. As a consequence, much research has been dedicated to analysing communication patterns, or protocols, among sets of agents in distributed systems. Examples of such research goals are to show the presence or absence of certain safety properties in a given system, to automate such analysis, and to guarantee the presence of desirable properties by construction. In this work we focus on analysing *choreographies* as global protocol specifications for asynchronously communicating agents. Choreographies have the benefit of guaranteeing certain safety properties by construction. We propose a new structure to compactly represent their behaviour based on partially ordered multisets (pomsets), which we call *branching pomsets*.

The use of choreographic languages is well-established [1, 2, 3, 4, 5, 6]. One of their typical uses is for reasoning statically over interaction properties, including deadlock freedom or the equivalence between global protocols and the parallel composition of local protocols for each agent. A more compact model of choreographies, as presented in this paper, could reduce the complexity of the analysis of protocols featuring both concurrency and choices. Specifically, in this work, we focus on the *realisability* property, i.e., whether a global specification of a protocol can be faithfully implemented in a distributed fashion in the first place. This problem has been well-studied in the last two decades in a variety of settings, with both synchronous and asynchronous communication [7, 2, 8, 9, 10].

A second typical use of choreographic languages is for generating code that facilitates the implementation of communication protocols. This includes skeleton code for concurrent code, generated behavioural types that can be used to type-check agents, and dedicated orchestrators that dictate how the agents can interact. This is beyond the scope of the present paper. However, in other recent work, we describe how to generate APIs using an

approach based on traditional sets of pomsets [11]. We are keen to extend it to take full advantage of the new model presented in this paper.

1.1. A first example

We use a simple example to further introduce choreographies and motivate our approach: the **review protocol**. This protocol governs the communications between three agents: Alice (**a**), Bob (**b**), and Carol (**c**). The former two agents are *reviewers*; the latter agent is the *editor* of a journal.

Suppose that Carol has received a new manuscript. The review protocol consists of two stages to determine if the paper can be accepted for publication. We explain both stages separately.

First stage: first round of reviewing. In the first stage, a request (**r**) to review the manuscript is communicated from Carol to both Alice and Bob (in parallel). Subsequently, ‘yes’ (**y**) or ‘no’ (**n**) is communicated back from both Alice and Bob to Carol to indicate whether or not they recommend acceptance (still in parallel). This first stage of the protocol can be represented as a choreography as follows, using the notation of this paper:

$$c_{\text{fst}} = (\mathbf{c} \rightarrow \mathbf{a} : \mathbf{r} ; (\mathbf{a} \rightarrow \mathbf{c} : \mathbf{y} + \mathbf{a} \rightarrow \mathbf{c} : \mathbf{n})) \parallel (\mathbf{c} \rightarrow \mathbf{b} : \mathbf{r} ; (\mathbf{b} \rightarrow \mathbf{c} : \mathbf{y} + \mathbf{b} \rightarrow \mathbf{c} : \mathbf{n}))$$

Here ‘ $\mathbf{c} \rightarrow \mathbf{a} : \mathbf{r}$ ’ denotes an asynchronous communication from **c** to **a** of a message of type **r**, ‘;’ denotes sequential composition, ‘ \parallel ’ denotes parallel composition and ‘+’ denotes free (i.e., unguarded) nondeterministic choice.

Second stage: optional second round. After the first stage, Carol may send Alice and Bob a request for a second review (e.g., after the manuscript has been revised). Alternatively, Carol may choose not to request a second review, for instance, if both Alice and Bob recommended ‘acceptance with minor revisions’ in the first round. If Carol sends a second request, she sends Alice and Bob a message ‘thanks’ (**t**) after receiving their reviews to thank them for their work and signal that their part is done. If Carol does not send a second request, she still sends the message **t** for the same reasons.

Formally, the second stage of the protocol can be interpreted in at least two ways: after sending a review request, Carol could either (1) wait for both replies before sending **t** to both Alice and Bob, or (2) send **t** to Alice as soon as she receives her reply, without waiting for Bob’s, and vice-versa. From now on we will refer to the first interpretation as ‘strict’ and to the second as

Table 1: Formal semantics of choreographies and the corresponding number of states (state-based models) and events (event-based models and this paper)

	state-based models	event-based models	this paper
typical example	LTS	pomset	branching pomset
representation of choice (+)	linear	exponential	linear
representation of parallelism ()	exponential	linear	linear

‘lenient’. Any mention of the review protocol without additional qualifiers refers to the lenient interpretation.

The strict interpretation can be represented as a choreography as follows, where $\mathbf{1}$ denotes succesful termination:

$$c_{\text{snd}}^{\text{strict}} = (c_{\text{fst}} + \mathbf{1}) ; (\textcolor{blue}{c} \rightarrow \textcolor{red}{a} : \textcolor{blue}{t} \parallel \textcolor{blue}{c} \rightarrow \textcolor{blue}{b} : \textcolor{blue}{t})$$

The lenient interpretation requires a more sophisticated replacement for the outermost sequential composition. It can be represented only by distributing the communication of the ‘thanks’ messages over $+$ (which duplicates the communications, resulting in a larger expression) and \parallel (inside c_{fst}), as follows:

$$c_{\text{snd}}^{\text{lenient}} = ((\textcolor{blue}{c} \rightarrow \textcolor{red}{a} : \textcolor{red}{r} ; (\textcolor{blue}{a} \rightarrow \textcolor{green}{c} : \textcolor{green}{y} + \textcolor{blue}{a} \rightarrow \textcolor{green}{c} : \textcolor{green}{n}) ; \textcolor{blue}{c} \rightarrow \textcolor{blue}{a} : \textcolor{blue}{t}) \parallel (\textcolor{blue}{c} \rightarrow \textcolor{blue}{b} : \textcolor{red}{r} ; (\textcolor{blue}{b} \rightarrow \textcolor{green}{c} : \textcolor{green}{y} + \textcolor{blue}{b} \rightarrow \textcolor{green}{c} : \textcolor{green}{n}) ; \textcolor{blue}{c} \rightarrow \textcolor{blue}{b} : \textcolor{blue}{t})) \\ + (\textcolor{blue}{c} \rightarrow \textcolor{blue}{a} : \textcolor{blue}{t} \parallel \textcolor{blue}{c} \rightarrow \textcolor{blue}{b} : \textcolor{blue}{t})$$

1.2. Problem: How to represent both choice and parallelism compactly

There are two major approaches to formalise the semantics of choreographies in the literature: *state-based models* and *event-based models*. Table 1 summarises the trade-off (for as far relevant to this paper).

State-based models. State-based models are good to represent choice (linear), but bad to represent parallelism (exponential). A typical such model is the *LTS semantics* of global types in the multiparty session types literature [12].

For instance, to demonstrate the explosion of states in the presence of parallel communications, the left of Figure 1 shows part of the LTS for $c_{\text{snd}}^{\text{strict}}$, whose full LTS consists of 44 states; the full LTS for the lenient version (i.e., more concurrent) consists of 64 states. In these LTSs, $\textcolor{blue}{ca}!\textcolor{red}{r}$ denotes a sending action from $\textcolor{blue}{c}$ to $\textcolor{blue}{a}$ with a message of type $\textcolor{red}{r}$, and $\textcolor{blue}{ca}?\textcolor{red}{r}$ denotes the dual receiving action.

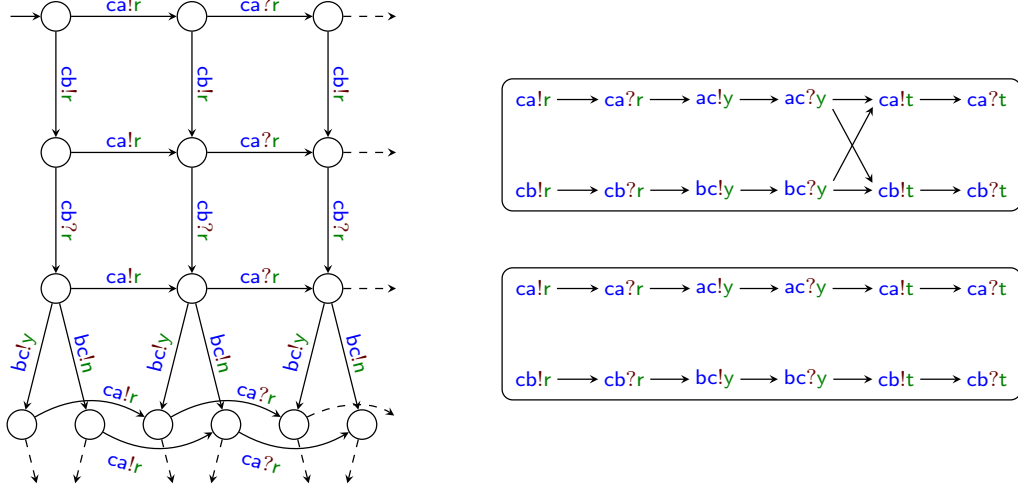


Figure 1: Part of a finite state machine (left) for the second stage of the strict review protocol, and two pomsets (right) representing one case of the second stage of, respectively, the strict (upper) and lenient (lower) review protocol.

Event-based models. Event-based models are bad to represent choice (exponential), but good to represent parallelism (linear). A typical such model is the *pomset semantics* of g-choreographies [13].

For instance, to demonstrate the non-explosion of events in the presence of parallelism (unlike the LTS semantics), the upper right of Figure 1 shows a graphical pomset representation of the special case of $\mathcal{C}_{\text{snd}}^{\text{strict}}$ where Carol sends a review request and both Alice and Bob reply y . The pomset contains 12 events (vs. 33 states in the LTS for this special case), whose labels are shown. The arrows represent the partial order between events: an event precedes, i.e., must occur before, any other event to which it has an outgoing arrow, either directly or transitively. In this example, the event with label $ac?y$ precedes the events with labels $ca!t$ and $cb!t$ directly and the events with labels $ca?t$ and $cb?t$ transitively. However, it is independent of the event with label $bc?y$ and those preceding it.¹ In general, the pomset grows

¹The synchronisation point between the two parallel branches is clearly represented in the upper pomset by means of the arrows from $ac?y$ and $bc?y$ to respectively $cb!t$ and $ca!t$. By removing these arrows, we obtain the analogous case in the lenient review protocol,

linearly with 6 events for each additional reviewer (i.e., with n reviewers, there are $6n$ events vs. $5^n + 3^n - 1$ states for this special case).

In contrast, to explain the explosion of events in the presence of choice, we note that the pomsets in Figure 1 only represent a single special case of the review protocol, namely that in which both reviewers recommend acceptance. Choices need to be represented as *sets* of pomsets: one for every possible combination of branches. Because of this, for the second stage of the review protocol we need $2 \times 2 + 1$ distinct pomsets: one for each combination of acceptance and rejection, plus one for the case where no review is requested. In general, while the size of each pomset only grows linearly with the number of reviewers, the *number* of pomsets (hence the total number of events) grows with roughly a factor 2 for each additional reviewer (i.e., with n reviewers, there are $2^n + 1$ pomsets in the set).

The problem. As summarised in Table 1, and explained above, neither state-based models nor event-based models offer a compact representation of both choice and parallelism. This is a problem, as many protocols mix these features (e.g., the review protocol). The aim of this paper is to offer a solution.

1.3. This paper

Contribution. This paper proposes an extension to pomsets, named *branching pomsets*, or *BPs* for short, with a branching structure that can compactly represent choices. In a nutshell, a BP initially contains all branches of choices, and discards non-chosen branches when firing events that require resolving a choice. For instance, the full behaviour of $c_{\text{snd}}^{\text{lenient}}$ is depicted as a BP in Figure 2, where each white box (except for the outermost one) represents one branch of a choice, while the choice itself is represented by the enclosing blue box. Each additional reviewer would expand the BP by just eight events and a single choice. While we initially introduced BPs specifically to model and study choreographies, we now define them as a generic model for concurrency and study them as such in the first half, before moving on to the use case of choreographies in the second half.

The concept of BP and the way we use it are reminiscent of event structures [14] and their recent usage in the context of multiparty session types [15]. Event structures were introduced in the 80s as a generalisation of posets with

shown in the lower pomset in the same figure.

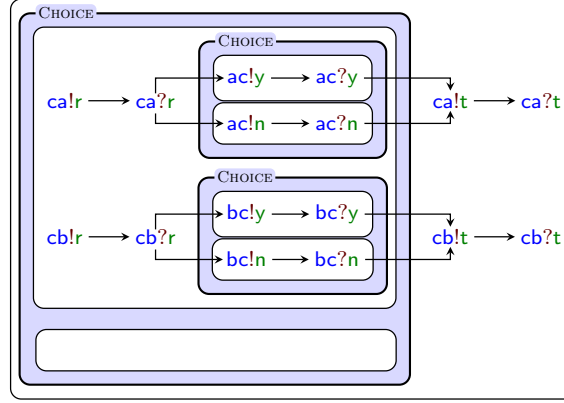


Figure 2: A BP for the second stage of the review protocol. We note that in this particular example, there are no two events with the same label. Thus, this pomset is actually a poset. However, if we had also represented the first round of the review, then all event labels except for the last four would have appeared twice in the resulting pomset.

branching, and similarly labelled event structures as a generalisation of pomsets. The main difference with BPs is in the added choice mechanism; in event structures this typically consists of a conflict relation, where two conflicting events may not occur together in the same execution. We give a thorough comparison between BPs and several classes of event structures in Section 3.

To aid in the understanding of BPs and their semantics, we provide a prototype tool to visualise and execute them, available at <https://lmf.di.uminho.pt/b-pomset/>. All the examples provided in the paper are predefined in the tool, such as $c_{\text{snd}}^{\text{lenient}}$ and $c_{\text{snd}}^{\text{strict}}$; their definitions in the remainder of the paper contain hyperlinks that open the tool with the specific example. We discuss the tool in more detail in Section 6.

Outline. The core contribution of this paper is the extension of pomsets with a branching structure, named BPs, in Section 2. We then explore this model in the following ways:

- In Section 3 we compare the expressiveness of BPs with that of several classes of **event structures (ESs)**. Specifically, we show that BPs define an interesting new class of behaviour: they are incomparable with extended bundle ESs and with growing and shrinking causality ESs. We conjecture that they describe a proper subset of a variant of dynamic causality ESs and we prove their inclusion in ESs for resolvable conflict.

- In [Section 4](#) we provide an **encoding from a choreographic language** into BPs and prove that the operational semantics of a choreography are equivalent (bisimilar) to those of its encoding as BP. Consequently, any static analysis of properties of a choreography can also be performed on the corresponding BP. This yields two main advantages:
 - As Guanciale and Tuosto argue in a recent paper [\[13\]](#), pomsets are syntax-oblivious. So are BPs. As a result, the analysis on BPs makes no assumptions on syntax and is applicable to a wide range of languages, as long as they can be encoded as BPs.
 - Compared to automata and traditional sets of pomsets, BPs supply additional structure in respectively concurrency and choices, yielding a more compact model. This structure makes it easier to reason over combinations of concurrency and choices, providing opportunities for more efficient analysis of choreographies featuring both.
- In [Section 5](#) we define **structural well-formedness properties** on BPs, inspired by multiparty session types (MPST) [\[3\]](#), and prove that they ensure realisability of the corresponding protocol. This approach sacrifices completeness for speed: the properties are easy to verify and ensure realisability, but there exist many realisable protocols which are nonetheless not well-formed. By defining these properties on BPs, they are not bound to the syntax of MPST. Consequently, verifying the conditions is slightly more complex as we can no longer take advantage of this syntax, but our results are applicable to any choreographic language which can be encoded as BPs. Furthermore, as BPs are a more generic model than global types in MPST, it may be easier to further generalise properties on BPs than on global types.
- In [Section 6](#) we describe our **prototype implementation**, both from a user and a developer’s perspective. The former explains how to use our tool to encode choreographies into BPs and how to analyse BPs using the techniques described in this paper. The latter explains how the well-formedness properties described in [Section 5.2](#) are realised by our tool, providing insights over its complexity.

We discuss related work in [Section 7](#). Finally, [Section 8](#) presents our conclusions and a brief discussion about future work.

This paper is an extended version of the paper with Guillermina Cledou, presented at ICE 2022 [\[16\]](#). We have joined it with the later FACS 2022 paper about realisability of BPs [\[17\]](#), which constitutes [Section 5](#) and parts of [Sections 7](#) and [8](#). The comparison with event structures ([Section 3](#)) and the description and analysis of the tool ([Section 6](#)) are new altogether.

2. Branching pomsets

In this section, we formally define the syntax and semantics of branching pomsets (BPs).

A partially ordered multiset [\[18\]](#), or pomset for short, consists of a set of nodes (events) E , a labelling function λ mapping events to a set of labels (e.g., send and receive actions), and a partial order \leq defining causal dependencies between pairs of events (i.e., an event can only fire if all events preceding it in the partial order have already fired). Its behaviour (or language) is the set of all traces labelling sequences of its events that abide by \leq .

Example 1. For the lower pomset in [Figure 1](#), $E = \{e_1, \dots, e_{12}\}$, λ is such that e_1, \dots, e_{12} map to respectively $\text{ca!r}, \text{ca?r}, \text{ac!y}, \text{ac?y}, \text{ca!t}, \text{ca?t}, \text{cb!r}, \text{cb?r}, \text{bc!y}, \text{bc?y}, \text{cb!t}, \text{cb?t}$ and $\leq = \{(e_i, e_j) \mid i \leq j \wedge (i, j \in \{1, \dots, 6\} \vee i, j \in \{7, \dots, 12\})\}$. Its behaviour consists of all the interleavings of $\text{ca!r}; \text{ca?r}; \text{ac!y}; \text{ac?y}; \text{ca!t}; \text{ca?t}$ and $\text{cb!r}; \text{cb?r}; \text{bc!y}; \text{bc?y}; \text{cb!t}; \text{cb?t}$.

As noted in [Section 1](#), however, there is no explicit representation of choices in pomsets, and they are represented only implicitly as a set of possible pomsets. We tackle this by extending pomsets with an explicit representation of choices: a branching structure on events.

2.1. Syntax

The general idea of a branching pomset is that all possible events are initially part of it, but some are defined as being part of a choice, depicted in [Figure 2](#) as choice boxes containing branches. The branching structure does not interrupt the partial order and all events still participate in it, as shown in the example, where arrows point both into and out of the branches of the choice. Nested choices are supported as well.

Formally, the branching structure of a BP is a tree whose leaves are events and whose inner nodes represent a structure of (possibly nested) choices and

branches. It is defined below with root node \mathcal{B} , whose children \mathcal{C} are either a single event e or a binary choice node with children (branches) $\mathcal{B}_1, \mathcal{B}_2$.

$$\begin{aligned}\mathcal{B} &::= \{\mathcal{C}_1, \dots, \mathcal{C}_n\} \\ \mathcal{C} &::= e \mid \{\mathcal{B}_1, \mathcal{B}_2\}\end{aligned}$$

Example 2. For the BP in Figure 2, $E = \{e_1, \dots, e_{16}\}$ and λ is such that e_1, \dots, e_{16} map to respectively $\text{ca!r}, \text{ca?r}, \text{ac!y}, \text{ac?y}, \text{ac!n}, \text{ac?n}, \text{ca!t}, \text{ca?t}, \text{cb!r}, \text{cb?r}, \text{bc!y}, \text{bc?y}, \text{bc!n}, \text{bc?n}, \text{cb!t}, \text{cb?t}$. The novelty with respect to pomsets is the branching structure $\mathcal{B} = \{e_7, e_8, e_{15}, e_{16}, \mathcal{C}_c\}$, where $\mathcal{C}_c = \{\{e_1, e_2, e_9, e_{10}, \mathcal{C}_a, \mathcal{C}_b\}, \emptyset\}$, $\mathcal{C}_a = \{\{e_3, e_4\}, \{e_5, e_6\}\}$ and $\mathcal{C}_b = \{\{e_{11}, e_{12}\}, \{e_{13}, e_{14}\}\}$ are the choices respectively by Carol, Alice and Bob.

We write $\mathcal{B}_1 \preceq \mathcal{B}_2$ if \mathcal{B}_1 is a subtree of \mathcal{B}_2 and $\mathcal{B}_1 \succ \mathcal{B}_2$ if \mathcal{B}_1 is a strict subtree of \mathcal{B}_2 , i.e., if $\mathcal{B}_1 \preceq \mathcal{B}_2$ and $\mathcal{B}_1 \neq \mathcal{B}_2$ ². We use the same notation for nodes \mathcal{C} , events e (a special case of \mathcal{C}) and combinations of all the aforementioned. Formally, the subtree relation is defined below, where \mathcal{N} can be either a \mathcal{B} or a \mathcal{C} (and thus also a singleton e).

$$\frac{}{\mathcal{N} \preceq \mathcal{N}} \quad \frac{\mathcal{N}_1 \in \mathcal{N}_2}{\mathcal{N}_1 \preceq \mathcal{N}_2} \quad \frac{\mathcal{N}_1 \preceq \mathcal{N}_2 \quad \mathcal{N}_2 \preceq \mathcal{N}_3}{\mathcal{N}_1 \preceq \mathcal{N}_3} \quad \frac{\mathcal{N}_1 \preceq \mathcal{N}_2 \quad \mathcal{N}_1 \neq \mathcal{N}_2}{\mathcal{N}_1 \succ \mathcal{N}_2}$$

Branching pomsets themselves are then formally defined below.

Definition 3 (Branching pomset). A branching pomset (BP) is a four-tuple $R = \langle E, \preceq, \lambda, \mathcal{B} \rangle$, where E is a set of events, $\preceq \subseteq E \times E$ is the causality relation such that \preceq^* (the reflexive and transitive closure of \preceq) is a partial order on events³, $\lambda : E \mapsto \mathcal{L}$ is a labelling function assigning to every event a label in some labelling set \mathcal{L} , and \mathcal{B} is a branching structure such that the set of leaves of \mathcal{B} is E and no event in E occurs in \mathcal{B} more than once.

We note that, in contrast to traditional pomsets, the causality relation \preceq is not necessarily a partial order. We briefly discuss this in Section 8. We use $R.E$, $R.\preceq$, $R.\lambda$ and $R.\mathcal{B}$ to refer to the components of R . We generally omit the prefix if doing so causes no confusion. We also write $e_1 \prec e_2$ if $e_1 \preceq e_2$ and $e_1 \neq e_2$. We use the following terminology:

²We use a different notation than in the original papers and their technical reports, where we used \preceq and \prec for the subtree relation.

³We use a different notation than in the original papers and their technical reports, where we used \leq for the causality relation. This is to make clearer that \preceq is not necessarily a partial order.

- Event e is *minimal* if $e' \not\prec e$ for all $e' \in R.E$ (i.e., there exists no other event e' that precedes e).
- Event e is *active* if $e \in R.B$ (i.e., e is not part of a choice).
- Event e is *enabled* if it is both active and minimal; we write $en(R)$ to denote the set of enabled events.
- Events e_1 and e_2 are *causally ordered* if either $e_1 \preceq e_2$ or $e_2 \preceq e_1$.
- Events e_1 and e_2 are *mutually exclusive* if there exists some $\mathcal{C} = \{\mathcal{B}_1, \mathcal{B}_2\} \succ R.B$ such that $e_1 \succ \mathcal{B}_1$ and $e_2 \succ \mathcal{B}_2$.

2.2. Semantics

Informally, every execution step of a BP R , in which an event e is fired, is brought about in three steps:

1. First, R is optionally *refined* to a “sub-BP” R' by resolving zero (i.e., $R = R'$), one, or more choices. Each resolution is done by replacing a choice $\{\{\mathcal{B}_1, \mathcal{B}_2\}\}$ at any level of the branching structure with one of its branches \mathcal{B}_i , thereby discarding the other branch \mathcal{B}_j . We note that this same idea governs the operational semantics of many existing languages, too. For instance, in process calculi, if P can reduce to P' , then also $P + Q$ (i.e., free choice between P and Q) can reduce to P' , thus resolving the choice and discarding Q .
2. Second, an *enabling* is sought. If an event e in R' is enabled and, additionally, e is disabled in every refinement R'' of R that is larger than R' (i.e., fewer choices are resolved in R'' than in R'), then refining R to R' is said to be an *enabling* of e . In other words, R' is the largest sub-BP of R in which e is enabled (i.e., the smallest number of choices are resolved to enable e). If $R = R'$, then zero choices were resolved in the first refinement step. In contrast, if $R \neq R'$, then one or more choices were resolved to enable e : either because e was a minimal event of a branch in R that was chosen in R' (so e also became active), or because e was active in R and causally ordered after events in a branch in R that was discarded in R' (so e also became minimal).
3. Third, R is *reduced* by firing e . The resulting BP is R' (the chosen refinement of R) without e .

$$\begin{array}{c}
\overline{\mathcal{B} \sqsupseteq \mathcal{B}}[\text{REFL}] \quad \frac{\mathcal{B} \sqsupseteq \mathcal{B}' \sqsupseteq \mathcal{B}''}{\mathcal{B} \sqsupseteq \mathcal{B}''}[\text{TRANS}] \quad \frac{i \in \{1, 2\} \quad \{\mathcal{B}_1, \mathcal{B}_2\} \notin \mathcal{B}}{\{\{\mathcal{B}_1, \mathcal{B}_2\}\} \cup \mathcal{B} \sqsupseteq \mathcal{B}_i \cup \mathcal{B}}[\text{CHOICE}] \\
\frac{\mathcal{B}_1 \sqsupseteq \mathcal{B}'_1 \quad \mathcal{B}_2 \sqsupseteq \mathcal{B}'_2 \quad \{\mathcal{B}_1, \mathcal{B}_2\} \notin \mathcal{B}}{\{\{\mathcal{B}_1, \mathcal{B}_2\}\} \cup \mathcal{B} \sqsupseteq \{\{\mathcal{B}'_1, \mathcal{B}'_2\}\} \cup \mathcal{B}}[\text{CONGR}] \quad \frac{R.\mathcal{B} \sqsupseteq \mathcal{B}'}{R \sqsupseteq R|_{\mathcal{B}'}}[\text{LIFT}]
\end{array}$$

(a) Refinement rules.

$$\begin{array}{c}
\frac{R \sqsupseteq R' \quad e \in \text{en}(R') \quad \forall R'' : R \sqsupseteq R'' \sqsupset R' \Rightarrow e \notin \text{en}(R'')}{R \xrightarrow{\check{e}} R'}[\text{ENABLE}] \\
\frac{R \xrightarrow{\check{e}} R'}{R \xrightarrow{e} R' - e}[\text{REDUCE1}] \quad \frac{R \xrightarrow{e} R'}{R \xrightarrow{\lambda(e)} R'}[\text{REDUCE2}] \quad \frac{R.\mathcal{B} \sqsupseteq \emptyset}{R \downarrow}[\text{TERMINATE}]
\end{array}$$

(b) Enabling, reduction, and termination rules.

$$\begin{aligned}
\langle E, \preceq, \lambda, \mathcal{B} \rangle|_{\mathcal{B}'} &= \langle E', \preceq \cap (E' \times E'), \lambda \cap (E' \times \mathcal{L}), \mathcal{B}' \rangle, \\
&\text{where } E' = \{e \mid e \succ \mathcal{B}'\} \\
\text{en}(R) &= \{e \in R.E \mid e \in R.\mathcal{B} \wedge \nexists e' \in R.E : e' \prec e\} \\
\hat{e} - e &= \hat{e} \\
\{\mathcal{C}_1, \dots, \mathcal{C}_n\} - e &= \begin{cases} \{\mathcal{C}_1, \dots, \mathcal{C}_{i-1}, \mathcal{C}_{i+1}, \dots, \mathcal{C}_n\} & \text{if } \mathcal{C}_i = e \\ \{\mathcal{C}_1 - e, \dots, \mathcal{C}_n - e\} & \text{otherwise} \end{cases} \\
\{\mathcal{B}_1, \mathcal{B}_2\} - e &= \{\mathcal{B}_1 - e, \mathcal{B}_2 - e\} \\
R - e &= R|_{R.\mathcal{B} - e}
\end{aligned}$$

(c) Operations on BPs.

Figure 3: Semantics of BPs.

The empty BP cannot perform execution steps; it is said to *terminate*. Any BP which can refine to the empty BP is able to terminate.

Formally, execution and termination are defined through relations:

- **Refinement:** A branching structure \mathcal{B} can refine to \mathcal{B}' , written $\mathcal{B} \sqsupseteq \mathcal{B}'$. We write $\mathcal{B} \sqsubset \mathcal{B}'$ to specify that $\mathcal{B} \neq \mathcal{B}'$. The refinement rules are

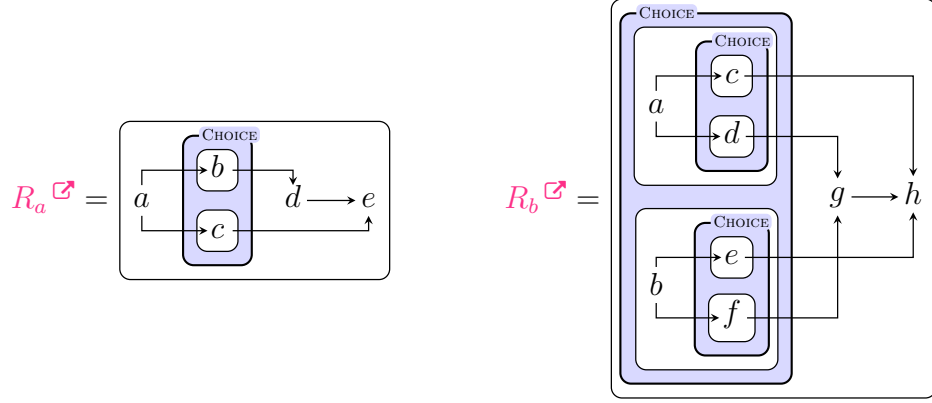


Figure 4: Two BPs.

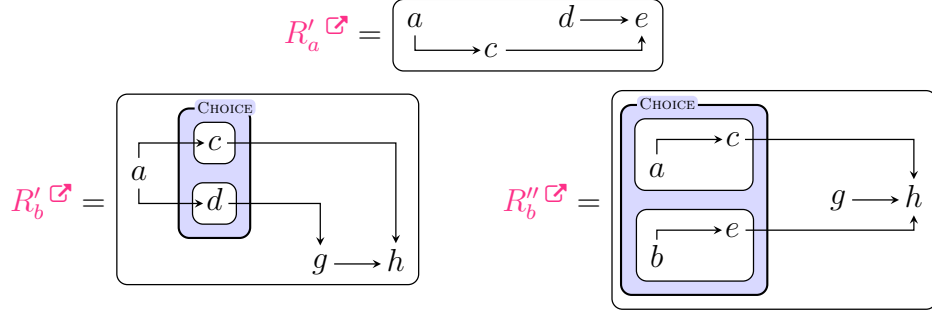


Figure 5: Refinements of R_a, R_b from Figure 4. R'_a is obtained by applying CHOICE to R_a . R'_b is obtained by applying CHOICE to the outer choice of R_b . R''_b is obtained by applying CHOICE to both inner choices and CONGR to the outer choice of R_b .

formalised in Figure 3a. To illustrate these, we use BPs R_a, R_b in Figure 4. As the labels are irrelevant for these examples, we use a, \dots, h for both the events and their labels. Finally, we assume that the relation \preceq for each BP consists exactly of the arrows shown in the figures.

The first two rules, REFL and TRANS, are straightforward. The third rule, CHOICE, resolves choices: it states that we can replace a choice with one of its branches. This rule serves a dual purpose: by applying it to the outer choice of R_b we can discard its lower branch, after which we can fire a , which is then active; alternatively, by applying it to R_a we can discard the upper branch of the choice, after which we can fire d , which is then minimal. Recall that \preceq is not necessarily a partial order and that, in R_a , $a \preceq^* d$ but not $a \preceq d$. Consequently,

the transitive dependency from a to d does not carry over to R'_a . The fourth rule, CONGR, allows us to resolve nested choices (with CHOICE) without first having to resolve their outer choices. To make g minimal in R_b we could resolve the outer choice and one inner choice with CHOICE (and TRANS). However, we can also apply CHOICE to resolve both inner choices and then apply CONGR to the outer choice to update it without unnecessarily resolving it. Finally, the fifth rule, LIFT, overloads the refinement notation to also apply to BPs themselves: if $R.\mathcal{B}$ can refine to some \mathcal{B}' then R itself can refine to a derived BP with branching structure \mathcal{B}' , written $R|_{\mathcal{B}'}$ ⁴, whose events are restricted to those occurring in \mathcal{B}' and likewise for \preceq and λ . The refinements above then lead to respectively R'_b , R'_a and R''_b in Figure 5.

- **Enabling, reduction, and termination:** Figure 3b defines an enabling relation, two reduction relations, and a termination predicate.

- The first rule, ENABLE, defines the conditions for enabling an event e , written $R \xrightarrow{e} R'$: a BP R can enable e by refining to R' if e is enabled in R' ($e \in en(R')$), and if there is no other refinement R'' in between which already enables e .

Example 4. Let $R_a, R_b, R'_a, R'_b, R''_b$ be as in Figures 4 and 5. Then:

$$\begin{aligned} * R_a &\xrightarrow{\check{d}} R'_a \\ * R_b &\xrightarrow{\check{a}} R'_b \\ * R_b &\xrightarrow{\check{g}} R''_b \end{aligned}$$

As in the informal description, the enabling relation only discards the absolutely necessary: for example, in the BP R_a in Figure 4, we may discard the choice's upper branch to fire d , but not to fire a . Similarly, in the BP R_b in the same figure, we may discard the lower branch from both inner choices to fire g , but there is no need to also resolve the outer choice. In Figure 2, we can discard the outer choice's upper branch to fire the event labelled **ca!t**.

The refinement rules in Figure 3a act as structural rules, which do not fire any event but may exclude events (by discarding branches),

⁴We use a different notation than in the original papers and their technical reports, where this restriction is written $R[\mathcal{B}']$.

as opposed to the reduction rules in Figure 3b, which fire events and are therefore computational rules. In fact, refinements could also be seen as executing silent transitions to resolve choices, as in process algebras with an internal choice operator, although, traditionally, in process algebras only top-level choices can be resolved in this way.

- The second and third rules, REDUCE1 and REDUCE2, define the reduction relations. They state that, if R can enable e by refining to R' (through ENABLE), then it can fire e by reducing to $R' - e$, which is the BP obtained by removing e from R' (Figure 3c). This reduction is defined both on e 's label (REDUCE2) and on the event itself (REDUCE1), the latter for internal use in proofs since $\lambda(e)$ is typically not unique but e is.
- The fourth rule, TERMINATE, defines the termination predicate and simply states that a BP can terminate if its branching structure can reduce to the empty set.

3. Comparison with event structures

In this section, we study the expressive power of branching pomsets by comparing them with various classes of event structures.

Event structures (ESs) are a well-established model for concurrency, which bears a close relationship with both Petri nets and domain theory. Originally introduced by Nielsen, Plotkin and Winskel [19, 14], this model represents a concurrent system as a set of (possibly labelled) events together with some relations among them, which regulate their occurrence in computations. Typically, in a prime event structure (PES), the original and simplest form of ES, there are two relations on events: *causality* (meaning that one event must occur before the other), and *conflict* (meaning that two events cannot occur in the same computation).⁵

In their labelled version, PESs may be viewed as pomsets enriched with a conflict relation. This makes them conceptually very close to branching pomsets, therefore a comparison is in order.

⁵To be more precise, the simplest class of ESs is that of *elementary* ESs, where the conflict relation is empty [14]. Elementary ESs are obtained from *causal nets* by retaining only their events and dropping their conditions. In fact, elementary ESs are just posets of events, or, in their labelled version, pomsets of event labels.

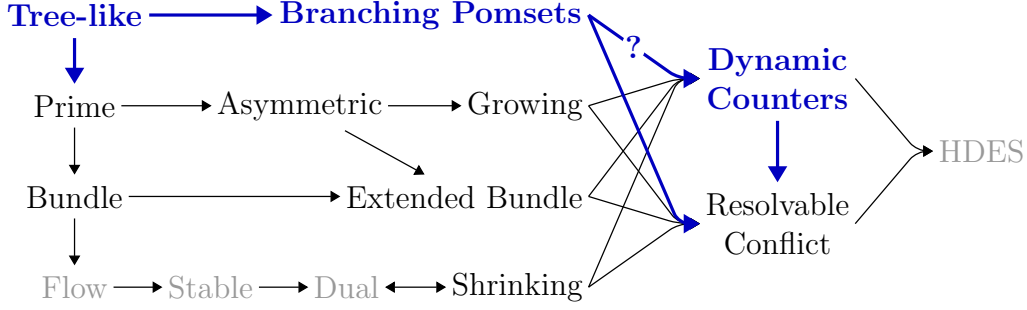


Figure 6: Landscape of event structures as extended from [20]. Branching pomsets are added in bold. Dynamic causality event structures are replaced by a variant with counters. Flow, stable, dual, and higher order dynamic causality event structures are faded to indicate that they are not discussed in detail.

Many variants of ESs have been studied in the last decades. We start by reviewing a number of them, referring for more details to the extensive overview given in the paper by Arbach et al. [20].

3.1. Event structure landscape

We first introduce the classes of ESs that are relevant to our study, namely those represented in Figure 6 (except for the faded ones, which are only included for completeness). We then uniformly define their semantics using the notion of *proving sequence*, from which the classical notion of *configuration* (a set of events that may have occurred at some stage of computation) may be immediately derived.

We start by considering the classes of *static* ESs, namely prime, bundle, asymmetric, and extended bundle ESs, where the relations on events are fixed once and for all. We then move to the classes of *dynamic* ESs, namely growing, shrinking and dynamic causality ESs, as well as ESs for resolvable conflict, where one of the two relations of causality and conflict may vary along execution.

3.1.1. Static event structures

The simplest class of ESs we consider, which is also the original one, is that of prime ESs.

Definition 5 (Prime Event Structure (PES) [19]). A prime event structure is a triple $S = \langle E, \#, \leq \rangle$, where E is a set of events, $\# \subseteq E \times E$ is a symmetric,

irreflexive relation called the *conflict* relation, and $\leq \subseteq E \times E$ is a partial order relation called the *causality* relation, satisfying the properties:

$$[e] = \{e' \mid e' \leq e\} \text{ is finite for all } e, e' \in E \quad (\text{finite causes})$$

$$e \# e' \wedge e' \leq e'' \implies e \# e'' \quad (\text{conflict hereditariness})$$

The condition of [conflict hereditariness](#) implies that the relations of conflict and causality are disjoint and that events do not have conflicting causes. Two events which are neither in conflict nor causally related are said to be *concurrent*. Two events which are not in conflict are said to be *consistent*. The axiom of [finite causes](#) forbids events with an infinite set of (consistent) causes, namely an event e such that $e_n \leq e$ for all $n \in \mathbb{N}$, thus ruling out also infinite regression (which could arise if we had also $e_{n+1} \leq e_n$ for all $n \in \mathbb{N}$).

Event structures were originally conceived as a system model and not as an algebraic model endowed with a set of constructors. However, since ESs were introduced roughly at the same time as the first process calculi CSP, CCS and ACP, they appeared as a natural candidate model⁶ to give semantics to process calculi. Because of their inability to represent disjunctive causality (i.e., events with multiple possible causes, only one of which needs to happen), PESs turned out to be too restrictive for that purpose. Typically, when two CCS processes are composed in parallel, some of their events (those carrying complementary labels) are allowed to synchronise giving rise to a new event, whose set of successor events should be the union of the sets of successors of the two original events, which can still occur independently. However, in a PES the conflict hereditariness condition prevents any sharing of the sets of successors, thus requiring their duplication after each synchronisation.

To overcome this problem, Winskel devised a more general class of ESs, called *stable event structures* [19], which accommodates disjunctive causality by replacing the causality relation \leq with an *enabling* relation between sets of events and events. Hence, the first ES semantics for CCS [21] was given in terms of stable ESs. However, since the enabling relation is a second-order relation on events, stable ESs lose the nice graphical representation offered by PESs. This observation motivated the subsequent introduction of two subclasses of stable ESs, respectively bundle ESs [22] and flow ESs [23],

⁶in their labelled version, where events have labels that represent process actions or communications.

which allow for disjunctive causality while retaining a graphical representation. Both these classes of ESs may be viewed as simple extensions of PESs, flow ESs being slightly more expressive than bundle ESs [24]. Here we only consider bundle ESs, which use a simpler enabling relation than stable ESs.

Definition 6 (Bundle Event Structure (BES) [22]). A bundle event structure is a triple $S = \langle E, \#, \rhd \rangle$, where E is a set of events, $\# \subseteq E \times E$ is the symmetric, irreflexive conflict relation and $\rhd \subseteq \mathcal{P}(E) \times E$ is the enabling relation, satisfying the property:

$$X \rhd e \implies \forall e_1 \neq e_2 \in X : e_1 \# e_2 \quad (\text{stability})$$

Intuitively, $X \rhd e$ means that *at least* one of the events in X needs to happen before e can happen. The condition of [stability](#) furthermore implies that *at most* one of the events in X needs to happen, as all the events in X must be pairwise in conflict with each other.

For both PESs and BESs, more general variants where conflict is not required to be symmetric have been proposed. These are called respectively asymmetric ESs and extended bundle ES. We recall their definitions below.

Definition 7 (Asymmetric Event Structure (AES) [25]). An asymmetric event structure is a triple $S = \langle E, \rightsquigarrow, \leq \rangle$, where E is a set of events, $\rightsquigarrow \subseteq E \times E$ is the *asymmetric conflict* relation, and $\leq \subseteq E \times E$ is the partially ordered *causality* relation, satisfying the following properties for all $e, e', e'' \in E$:

$$[e] = \{e' \mid e' \leq e\} \text{ is finite} \quad (\text{finite causes})$$

$$e < e' \implies e \rightsquigarrow e' \quad (1)$$

$$(e \rightsquigarrow e' \wedge e' < e'') \implies e \rightsquigarrow e'' \quad (2)$$

$$\rightsquigarrow \text{ is acyclic on } [e] \quad (3)$$

$$(\rightsquigarrow \text{ cyclic on } [e] \cup [e']) \implies e \rightsquigarrow e' \quad (4)$$

In the above definition, Condition 2 expresses hereditariness of asymmetric conflict, while Condition 3 rules out cycles of asymmetric conflict in the set of causes of an event. Since cycles can be self-cycles, this implies in particular that asymmetric conflict is irreflexive. As for Condition 4, it requires that any semantic conflict between two events which is due to their

co-occurrence on a cycle of asymmetric conflict be explicitly represented by an asymmetric conflict in both directions.

As explained in [25], the asymmetric conflict relation has two natural interpretations: $e \rightsquigarrow e'$ may be understood as (i) e' *disables* e , namely the occurrence of e' prevents the occurrence of e , or (ii) e (*weakly*) *precedes* e' , namely e occurs before e' in all executions where they both occur.

A similar disabling relation is used in extended bundle ESs.

Definition 8 (Extended Bundle Event Structure (EBES) [26]). An extended bundle event structure is a triple $S = \langle E, \rightsquigarrow, \succ \rangle$, where E is a set of events, $\rightsquigarrow \subseteq E \times E$ is the irreflexive *disabling* relation and $\succ \subseteq \mathcal{P}(E) \times E$ is the *enabling* relation, satisfying the following:

$$X \succ e \implies \forall e_1 \neq e_2 \in X : e_1 \rightsquigarrow e_2 \quad (\text{stability condition})$$

Here again, $e \rightsquigarrow e'$ should be read from right to left as e' *disables* e .

The construction of an EBES from an AES is similar to that of a BES from a PES. As such, AESs are included in EBESs.

3.1.2. Dynamic Event Structures

We review now the classes of dynamic ESs, where one of the two relations of causality and conflict may dynamically change along execution.

In [20], three new classes of ESs have been proposed, where the causality relation can be modified by effect of the occurrence of some other event: (1) *shrinking causality event structures (SESSs)*, where causal dependencies can be removed, (2) *growing causality event structures (GESs)*, where causal dependencies can be added, and (3) *dynamic causality event structures (DCESSs)*, where causal dependencies can be both added and removed.

Shrinking ESs extend rPESs (relaxed PESs where the conflict relation is not required to be hereditary⁷) by allowing the causality relation to decrease along execution.

Definition 9 (Shrinking Causality Event Structure (SES) [20]). A shrinking causality event structure is a quadruple $S = \langle E, \#, \rightarrow, \triangleleft \rangle$, where E is a set of events, $\# \subseteq E \times E$ is the symmetric, irreflexive *conflict* relation, $\rightarrow \subseteq E \times E$

⁷While the name “rPES” has been coined in [20], this PES variant had already been used to interpret a subset of CCS in [27].

is the *initial causality* relation and $\triangleleft \subseteq E \times E \times E$ is the *shrinking causality* relation satisfying the property:

$$e \triangleleft [e_1 \rightarrow e_2] \implies (e_1 \rightarrow e_2 \wedge e \notin \{e_1, e_2\}) \quad (\text{SC})$$

Note that SESs can model disjunctive causality. Indeed, the shrinking causality $e \triangleleft [e' \rightarrow e'']$ represents a situation where initially e' causes e'' , but this causality may be cancelled by the occurrence of e . Thus, if $e \triangleleft [e' \rightarrow e'']$ and $e' \triangleleft [e \rightarrow e'']$ and $e \# e'$, then e and e' are two conflicting causes of e'' . On the other hand, if $e \triangleleft [e' \rightarrow e'']$ and $e' \triangleleft [e \rightarrow e'']$ and $\neg(e \# e')$, then both e and e' may occur in the same computation, thus we can reach a state where all of e, e', e'' have occurred: in this state, we do not know which of e or e' has caused e'' (one of them must, since e'' cannot occur alone). This means that SESs are not stable. Moreover, SESs cannot model disabling: for instance the EBES with two events e and e' where $e \rightsquigarrow e'$ and $\rightsquigarrow = \emptyset$ cannot be simulated by a SES.

Dually, growing causality ESs extend rPESs by allowing the causality relation to increase along execution.

Definition 10 (Growing Causality Event Structure (GES) [20]). A growing causality event structure is a triple $S = \langle E, \rightarrow, \blacktriangleright \rangle$, where E is a set of events, $\rightarrow \subseteq E \times E$ is the *initial causality* relation and $\blacktriangleright \subseteq E \times E \times E$ is the *growing causality* relation satisfying the property:

$$e \blacktriangleright [e_1 \rightarrow e_2] \implies (\neg(e_1 \rightarrow e_2) \wedge e \notin \{e_1, e_2\}) \quad (\text{GC})$$

Note that conflict does not appear in [Theorem 10](#). This is because conflict may be simulated by mutual disabling, and disabling may be simulated by growing causality. The simulation of disabling as given in [20] assumes the existence of an “impossible” event e_{imp} ⁸ such that $e_{\text{imp}} \rightarrow e_{\text{imp}}$ (exploiting the fact that \rightarrow is not required to be irreflexive): then a conflict between e and e' may be modelled by setting $e \blacktriangleright [e_{\text{imp}} \rightarrow e']$ and $e' \blacktriangleright [e_{\text{imp}} \rightarrow e]$. In fact, since [Theorem 10](#) does not require $e_1 \neq e_2$ in Condition (GC), the conflict between e and e' may also be simulated more directly (see [20] again) by letting $e \blacktriangleright [e' \rightarrow e']$ and $e' \blacktriangleright [e \rightarrow e]$. Note finally that, unlike SESs, GESs cannot model disjunctive causality.

⁸The existence of impossible events, namely events that cannot occur in any computation, is a common feature of all classes of ESs except PESs. When a particular ES does not contain such events, it is said to be *full*.

Combining the features of shrinking and growing ESs, and adding some constraints on the interplay between shrinking and growing causality, we obtain dynamic causality ESs (DCESSs). In this paper, we consider a variant of DCESSs, with the same syntax but subtly different semantics. This is discussed when we formally define the semantics.

Definition 11 (Dynamic Causality Event Structure [20]). A dynamic causality event structure (DCESS) is a quadruple $S = \langle E, \rightarrow, \triangleleft, \blacktriangleright \rangle$, where E is a set of events, $\rightarrow \subseteq E \times E$ is the *initial causality* relation, and $\triangleleft, \blacktriangleright \subseteq E \times E \times E$ are respectively the *shrinking* and *growing causality* relations, such that:

$$\nexists e' \in E : e' \blacktriangleright [e_1 \rightarrow e_2] \wedge e \triangleleft [e_1 \rightarrow e_2] \implies e_1 \rightarrow e_2 \quad (5)$$

$$e \triangleleft [e_1 \rightarrow e_2] \implies e \notin \{e_1, e_2\} \quad (6)$$

$$\nexists e' \in E : e' \triangleleft [e_1 \rightarrow e_2] \wedge e \blacktriangleright [e_1 \rightarrow e_2] \implies \neg(e_1 \rightarrow e_2) \quad (7)$$

$$e \blacktriangleright [e_1 \rightarrow e_2] \implies e \notin \{e_1, e_2\} \quad (8)$$

$$e \blacktriangleright [e_1 \rightarrow e_2] \implies \neg(e \triangleleft [e_1 \rightarrow e_2]) \quad (9)$$

Again, conflict is omitted because it may be simulated by growing causality. Conditions 6 and 8 correspond to the second half of Conditions (SC) and (GC) respectively. Conditions 5 and 7 correspond to the first half of Conditions (SC) and (GC), accounting for the possibility that the other operator could add or subtract some causal dependencies. Finally, Condition 9 prevents an event from adding and dropping the same causal dependency.

The last class of dynamic ESs we shall consider is the following, where the conflict relation may be changed dynamically.

Definition 12 (Resolvable Conflict Event Structure (RCES) [28]). An event structure for *resolvable conflict* is a pair $S = \langle E, \vdash \rangle$, where E is a set of events and $\vdash \subseteq \mathcal{P}(E) \times \mathcal{P}(E)$ is the *enabling* relation.

We recall from [28] a simple example showing that RCESs can model resolvable conflicts, namely conflicts that disappear by effect of the occurrence of some event. Consider the structure $\langle E, \vdash \rangle$ where $E = \{a, b, c\}$, with $\{a\} \vdash \{b, c\}$ and $\emptyset \vdash X$ iff $X \subseteq E$ and $X \neq \{b, c\}$. This models an initial conflict between b and c , which can be resolved by the occurrence of a . We refer the reader to [28] for more examples, showing that RCESs are not stable. In fact, in [28] RCESs are shown to be able to represent any Petri net, a very strong expressiveness result.

3.1.3. Event Structure Semantics

The semantics of ESs is classically defined in terms of configurations. A *configuration* is a set of events that may have occurred at some stage of a computation. For all classes of ESs, we shall uniformly define a configuration to be a set of events enumerable as a *proving sequence* [19, 24, 26], namely a sequence of consistent events such that each event is “secured” - i.e., granted the possibility to occur - by the preceding ones. We will first introduce proving sequences for all classes of ESs, and then uniformly derive from them a notion of configuration and a reduction relation on configurations.

We start by introducing some terminology. A *trace* is a finite sequence of distinct events $t = e_1 \dots e_n$, where $n \geq 0$ and by convention $t = \varepsilon$ if $n = 0$. Given a trace $t = e_1 \dots e_n$, we denote by $t_i = e_1 \dots e_i$ its prefix of length i for every $i \leq n$, and by \bar{t} the set $\{e_1, \dots, e_n\}$ of events occurring in t . In particular, $t_n = t$, $t_0 = \varepsilon$ and $\bar{t}_0 = \emptyset$. A set of events is *consistent* if it is conflict-free.

Proving sequences are traces with two distinguishing properties: *consistency* of the underlying set of events, and *securing* for each of their events. Their formal definitions differ depending on the considered class of ESs.

Definition 13 (PES proving sequence [19]). Let $S = \langle E, \#, \leq \rangle$ be a prime event structure. A proving sequence of S is a trace $t = e_1 \dots e_n$ with $\bar{t} \subseteq E$, satisfying the properties:

$$\forall i, j \in [1, n] : \neg(e_i \# e_j) \quad (\text{consistency})$$

$$\forall i \in [1, n] : \forall e \in E : (e < e_i \implies e \in \overline{t_{i-1}}) \quad (\text{left-closure})$$

Definition 14 (AES proving sequence [25]). Let $S = \langle E, \rightsquigarrow, \leq \rangle$ be an asymmetric event structure. A proving sequence of S is a trace $t = e_1 \dots e_n$ with $\bar{t} \subseteq E$, satisfying the properties:

$$\forall i, j \in [1, n] : e_i \rightsquigarrow e_j \implies i < j \quad (\text{consistency})$$

$$\forall i \in [1, n] : \forall e \in E : (e < e_i \implies e \in \overline{t_{i-1}}) \quad (\text{left-closure})$$

For PESs and asymmetric ESs, which have a global causality relation \leq , securing amounts to *left-closure* with respect to \leq . For more expressive ESs such as BESs and EBESs, which allow for disjunctive causality and only recover a partial order of causality within individual computations, securing is defined as *consistent left-closure*, namely left-closure up to conflicts. In BESs and EBESs, this property is called *bundle satisfaction*.

Definition 15 (BES proving sequence [26]). Let $S = \langle E, \#, \succ \rangle$ be a bundle event structure. A proving sequence of S is a trace $t = e_1 \dots e_n$ with $\bar{t} \subseteq E$ satisfying the properties:

$$\forall i, j \in [1, n] : \neg(e_i \# e_j) \quad (\text{consistency})$$

$$\forall i \in [1, n] : \forall X \subseteq E : (X \succ e_i \implies X \cap \overline{t_{i-1}} \neq \emptyset) \quad (\text{bundle satisfaction})$$

Definition 16 (EBES proving sequence [26]). Let $S = \langle E, \rightsquigarrow, \succ \rangle$ be an extended bundle event structure. A proving sequence of S is a trace $t = e_1 \dots e_n$ with $\bar{t} \subseteq E$ satisfying the properties:

$$\forall i, j \in [1, n] : e_i \rightsquigarrow e_j \implies i < j \quad (\text{consistency})$$

$$\forall i \in [1, n] : \forall X \subseteq E : (X \succ e_i \implies X \cap \overline{t_{i-1}} \neq \emptyset) \quad (\text{bundle satisfaction})$$

For dynamic classes of ESs, the definition of proving sequence is more subtle, since it needs to account for the fact that the causes of every event in the sequence may have been modified by some earlier event in the sequence. The definitions we give below for SESs and GESs are slightly different in form, but semantically equivalent, to the ones given in [20].

Definition 17 (SES proving sequence [20]). Let $S = \langle E, \#, \rightarrow, \triangleleft \rangle$ be a shrinking causality event structure. A proving sequence of S is a trace $t = e_1 \dots e_n$ with $\bar{t} \subseteq E$ satisfying the properties:

$$\forall i, j \in [1, n] : \neg(e_i \# e_j) \quad (\text{consistency})$$

$$\begin{aligned} \forall i \in [1, n] : \forall e \in E : \\ e \rightarrow e_i \implies (e \in \overline{t_{i-1}} \vee \exists e' \in \overline{t_{i-1}} : e' \triangleleft [e \rightarrow e_i]) \end{aligned} \quad (\text{securing})$$

Informally, the securing property for SESs means that if e causes e_i initially ($e \rightarrow e_i$), then either e has indeed happened before e_i ($e \in \overline{t_{i-1}}$), or another event e' has happened that removed the causality ($\exists e' \in \overline{t_{i-1}} : e' \triangleleft [e \rightarrow e_i]$).

Definition 18 (GES proving sequence [20]). Let $S = \langle E, \rightarrow, \blacktriangleright \rangle$ be a growing causality event structure. A proving sequence of S is a trace $t = e_1 \dots e_n$ with $\bar{t} \subseteq E$ satisfying the property:

$$\forall i \in [1, n] : \forall e \in E : (e \rightarrow e_i \vee \exists e' \in \overline{t_{i-1}} : e' \blacktriangleright [e \rightarrow e_i]) \implies e \in \overline{t_{i-1}}$$

Informally, the securing property for GESs means that if either e causes e_i initially ($e \rightarrow e_i$), or if another event e_j has happened that added the causality ($e' \blacktriangleright [e \rightarrow e_i]$), then e has indeed happened before e_i ($e \in \overline{t_{i-1}}$).

Recall that DCESSs combine the power of SESs and GESs: they are essentially PESs whose causality relations can both shrink and grow as events happen in a computation. Relative to SESs and GESs, the key complication to define the semantics of DCESSs is that *the same causality* can be removed, added, removed again, etc., in the same computation. In contrast, in SESs (resp. GESs), once a causality is removed (resp. added), it remains absent (resp. present) forever. Thus, a more advanced bookkeeping mechanism is needed to account for the additions/removals of causalities to define proving sequences of DCESSs.

In [20] the operations for adding and dropping a cause are idempotent, in the sense that the executions of two successive additions (resp. removals) of the same causal dependency have the same effect as that of a single one. Here, we adopt a finer mechanism, which counts the number of additions and removals along an execution. Thus, our semantics for DCESSs is a variant of the original one proposed in [20], which we will call *dynamic causality event structures with counters* (DCCESs). One advantage of this new semantics is that it supports a simple definition of the transition relation on configurations, which does not require the pairing of configurations with an ordering as in [20]. More precisely, the intuition for a trace $t = e_1 \dots e_n$ to be a proving sequence of a DCCES is that for any event e_i in the trace, any cause of e_i which has been added more times than it has been dropped by events occurring in the prefix t_{i-1} does effectively cause e_i at this stage of computation, and therefore it should occur in the prefix t_{i-1} . The definition relies on some auxiliary multisets, which are variations (enriched with multiplicities) of the auxiliary sets used in [20]. Clearly DCCESs extend both SESs and GESs. The proof for EBESs given in [20] also holds for DCCESs, thus giving the inclusions shown in Figure 6.

In the following definition we use $e^{(k)} \in X$ to indicate that e occurs in the multiset X with multiplicity k (where in case X is a set we write $e \in X$ instead of $e^{(1)} \in X$), $\text{mult}(e, X)$ to denote the multiplicity of e in X , and $|X|$ to denote the cardinality of the set X .

Definition 19 (Set of initial causes, multisets of added and dropped causes). Let $S = \langle E, \rightarrow, \triangleleft, \blacktriangleright \rangle$ be a dynamic causality event structure with counters. Let $e \in E$ and $t = e_1 \dots e_n$ with $\bar{t} \subseteq E$. Then:

1. The *set of initial causes of e* is defined by $\text{ic}(e) = \{e' \mid (e', e) \in \rightarrow\}$;
2. The *multiset of dropped causes of e after t* is defined by $\text{dc}(e, t) = \{e'^{(k)} \mid k = |\{e'' \in \bar{t} \mid e'' \triangleleft (e', e)\}|\}$;
3. The *multiset of added causes of e after t* is defined by $\text{ac}(e, t) = \{e'^{(k)} \mid k = |\{e'' \in \bar{t} \mid e'' \blacktriangleright (e', e)\}|\}$.

Definition 20 (DCCES proving sequence). Let $S = \langle E, \rightarrow, \triangleleft, \blacktriangleright \rangle$ be a dynamic causality event structure with counters. A proving sequence of S is a trace $t = e_1 \dots e_n$ with $\bar{t} \subseteq E$ satisfying the property: $\forall i \in [1, n] : \forall e \in E :$

$$\text{mult}(e, \text{ic}(e_i)) + \text{mult}(e, \text{ac}(e_i, t_{i-1})) - \text{mult}(e, \text{dc}(e_i, t_{i-1})) \geq 1 \implies e \in \overline{t_{i-1}}$$

Finally, we define proving sequences for RCEs directly, in agreement with the RCEs semantics in [28]:

Definition 21 (RCEs proving sequence). Let $S = \langle E, \vdash \rangle$ be an event structure for resolvable conflict. A proving sequence of S is a trace $t = e_1 \dots e_n$ with $\bar{t} \subseteq E$ satisfying the property:

$$\forall i \in [1, n] : \forall Z \subseteq \bar{t}_i : \exists W \subseteq \overline{t_{i-1}} : W \vdash Z$$

Informally, the securing property for RCEs means that every subset of events Z at timestamp i must be enabled by a subset of events W at timestamp $i - 1$.

Having introduced the notion of proving sequence for each class of ESs, we may now uniformly define the notion of configuration and a transition system on configurations for all kinds of ESs.

Definition 22 (Configuration and transition system). Let S be any ES. A *configuration* of S is any set X such that $X = \bar{t}$ for some proving sequence t of S . The *set of configurations* of S is $C(S) = \{\bar{t} \mid t \text{ is a proving sequence of } S\}$. The *multi-step transition relation* $\Rightarrow_S \subseteq C(S) \times C(S)$ on configurations is then defined as: $X \Rightarrow_S Y$ if there exist proving sequences t_1 and t_2 of S such that $\bar{t}_1 = X$ and $\bar{t}_2 = Y$ and t_1 is a prefix of t_2 . We then derive the *single action transition relation* \mapsto_S as: $X \mapsto_S Y$ if $X \Rightarrow_S Y$ and $|Y| = |X| + 1$.

We state some simple properties of proving sequences, which will be used for proving [Theorem 25](#):

Lemma 23. *Let S be an ES of any of the aforementioned classes. Then:*

1. If t is a proving sequence of S , then any prefix t_i of t is also a proving sequence of S ;
2. Let t and t' be two proving sequences of S such that $\bar{t} = \bar{t}'$. If te is a proving sequence of S , then also $t'e$ is a proving sequence of S .⁹

We define a transition equivalence generically on any type of ES.

Definition 24 (ES transition equivalence [28]). Two ESs S_1 and S_2 are called *multi-step transition equivalent*, written $S_1 \simeq_{mt} S_2$, if $\models_{S_1} = \models_{S_2}$. Analogously, two ESs S_1 and S_2 are called *single action transition equivalent*, written $S_1 \simeq_t S_2$, if $\mapsto_{S_1} = \mapsto_{S_2}$.

We note that, since we have only defined single action semantics for BPs, we will only use the single action transition relation for ESs in the remainder. We will thus simply use *transition relation* and *transition relation equivalence*, without further qualifiers, to refer to the single action ones.

We conclude by showing that, using these semantics, any type of ES considered in this paper can be encoded as an RCES. In particular, DCCESs also represent a subset of RCESs, while DCEs and RCESs have been shown to be incomparable [20].

Theorem 25. *Let S be an ES of any of the aforementioned classes, with set of events E , and let \mapsto_S be the corresponding transition relation as given by Theorem 22. Then there exists an RCES $\hat{S} = \langle E, \vdash \rangle$ such that $\mapsto_{\hat{S}} = \mapsto_S$.*

Proof. First we use the proving sequences of S to define \vdash . For any proving sequence $t = e_1 \dots e_n$ of S , define

$$\vdash_t = \{(\overline{t_{i-1}}, Z_i) \mid Z_i \subseteq \bar{t}_i, i \in 1, \dots, n\}.$$

Notice that for any t , we have $(\emptyset, \bar{t}_1) \in \vdash_t$. Let now

$$\vdash = \bigcup \{\vdash_t \mid t \text{ is a proving sequence of } S\}.$$

We proceed to show that S and \hat{S} have the same sets of proving sequences.

⁹We note that this is true for any class of ESs in which the current configuration (i.e., the *set* of occurred events) wholly determines the current causal state (i.e., which events are enabled). In particular, this is true for DCCESs but not for DCEs, which is why Theorem 25 does not hold for the latter.

- Let $t = e_1 \dots e_n$ be a proving sequence of S . Then, for any $1 \leq i \leq n$ and for any $Z_i \subseteq \bar{t}_i$, we have $\bar{t}_{i-1} \vdash_t Z_i$. It follows from [Theorem 21](#) that t is then also a proving sequence of \hat{S} .
- We prove now that any proving sequence $t = e_1 \dots e_n$ of \hat{S} is also a proving sequence of S . We proceed by induction on the length n of t . For $n = 0$ we have $t = t_0 = \varepsilon$, which is a proving sequence of S by definition. Assume now that the statement holds for all proving sequences of length k , for $0 \leq k \leq n - 1$. We want to show that it holds also for $t = e_1 \dots e_n = t_{n-1}e_n$. By induction t_{n-1} is a proving sequence of S . Since t is a proving sequence of \hat{S} , by [Theorem 21](#) for all $Z \subseteq \bar{t}$ there exists some $W \subseteq \bar{t}_{n-1}$ such that $W \vdash Z$. In particular this holds for $Z = \bar{t}$. Note that, on the one hand, $W \subseteq \bar{t}_{n-1}$ implies that $|W| \leq n - 1$. On the other hand, the construction of \vdash implies that, if $W \vdash Z$, then $|Z| < |W|$. Since $|\bar{t}| = n$, it follows that if $W \vdash \bar{t}$ then it must be $|W| = n - 1$ and thus $W = \bar{t}_{n-1}$. In other words: $\bar{t}_{n-1} \vdash_{t'} \bar{t}$ for some proving sequence t' of S . Then, by definition of $\vdash_{t'}$, $\bar{t}_{n-1} = \bar{t}'_{n-1}$ and $\bar{t} \subseteq \bar{t}'_n$. Since $t = t_n$ and t_n and t'_n have the same length, it follows that $\bar{t} = \bar{t}'_n$. This means that $t'_n = t'_{n-1}e_n$. Now, since t'_n and t'_{n-1} are prefixes of t' , by [Theorem 23\(1\)](#) they are also proving sequences of S . Then we may use [Theorem 23\(2\)](#) to conclude that $t = t_n = t_{n-1}e_n$ is a proving sequence of S .

Since S and \hat{S} have the same proving sequences, it directly follows from [Theorem 22](#) that $\mapsto_S = \mapsto_{\hat{S}}$. \square

3.2. Comparing event structures and branching pomsets – Overview

We now compare branching pomsets with the various classes of ESs previously reviewed. We establish a number of relative expressiveness results, summarised in [Figure 7](#), where we complete the initial picture of [Figure 6](#) with dashed red lines representing the non-inclusion of one model into another. Non-inclusion results are proved by providing counterexamples. The inclusion of tree-like BPs into PESs is proved in [Theorem 31](#). The inclusion of general BPs into RCESSs is proved in [Theorem 36](#). The inclusion of general BPs into DCESSs is work in progress and for now remains a conjecture.

To conduct this comparison, we first need to introduce configurations also on branching pomsets, as well as a transition relation between them.

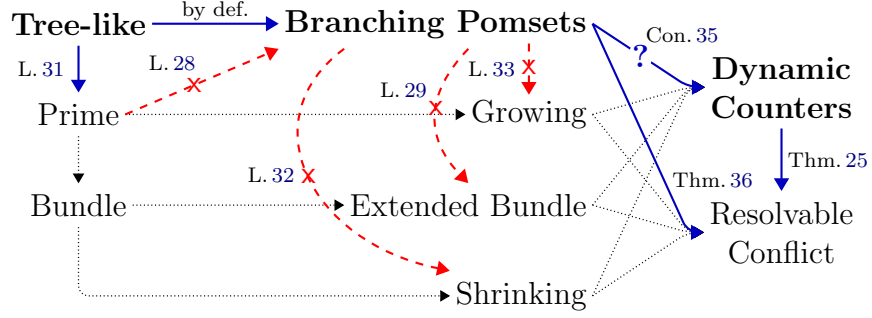


Figure 7: Summary of the core results proven in this section; solid blue arrows \longrightarrow represent strict inclusion, and dashed lines $- \times - \blacktriangleright$ represent non-inclusion. A question mark indicates a conjecture. Labels are active pointers to Lemmas, Theorems and Conjectures.

Definition 26 (Configuration). Let $R = \langle E, \preceq, \lambda, \mathcal{B} \rangle$ be a branching pomset. Then $X \subseteq E$ is a configuration of R if there exists some trace $t = e_1 \dots e_n$ such that $R \xrightarrow{t}^* R'$ and $\bar{t} = X$.

Just as for ESs, let $C(R)$ be the set of configurations of R .

Definition 27 (Transition system). Let $R = \langle E, \preceq, \lambda, \mathcal{B} \rangle$ be a branching pomset and let $X \subseteq E$. Then the transition relation $\mapsto_R \subseteq C(R) \times C(R)$ is defined as follows: $X \mapsto_R X \cup \{e\}$ if $R \xrightarrow{t}^* R' \xrightarrow{e}$ for some $t = e_1 \dots e_n$ such that $\bar{t} = X$.

We may now proceed to prove the results and discuss the conjectures corresponding to the solid blue arrows and dashed red arrows in Figure 7.

3.3. Comparing event structures and branching pomsets – Static models

Our first result states that the class of prime ESs is not included in that of BPs. Since prime ESs are the simplest class of ESs (static and dynamic alike), extended by all the others, this implies that no class of ESs is included in that of BPs.

Lemma 28 ($PES \not\subseteq BP$). *There exists a prime event structure S for which there does not exist a branching pomset R such that $C(S) = C(R)$.*

Proof. Let $S = \langle E, \#, \leq \rangle$ be the PES in Figure 8a, where $E = \{a, b, c, d\}$, $a \# b, a \# d, c \# d$ and $a \leq a, b \leq b, c \leq c, d \leq d$. Assume, for the sake of contradiction, that there exists some BP $R = \langle E, \preceq, \lambda, \mathcal{B} \rangle$ with the same set of configurations, namely $C(R) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{d\}, \{a, c\}, \{b, c\}, \{b, d\}\}$.

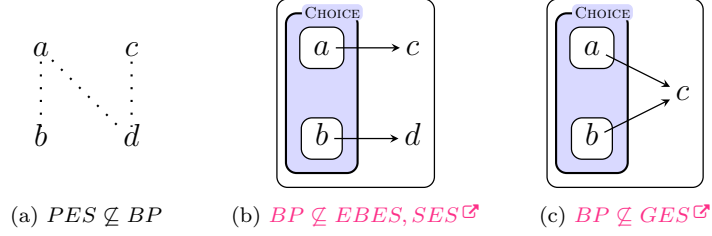


Figure 8: Counterexamples

Then, $a \# b$ implies that there exists some choice $\mathcal{C}_1 = \{\mathcal{B}_1, \mathcal{B}_2\}$ such that $a \succ \mathcal{B}_1$ and $b \succ \mathcal{B}_2$. Analogously, $c \# d$ implies that there exists $\mathcal{C}_2 = \{\mathcal{B}_3, \mathcal{B}_4\}$ such that $c \succ \mathcal{B}_3$ and $d \succ \mathcal{B}_4$.

Now there are four possible ways to relate \mathcal{C}_1 and \mathcal{C}_2 in \mathcal{B} . We show that all of them lead to a contradiction.

1. Suppose \mathcal{C}_1 and \mathcal{C}_2 are concurrent choices: $\mathcal{B} = \{\mathcal{C}_1, \mathcal{C}_2\}$. Then, if $a \preceq d$ it follows that $R \xrightarrow{ad}^* R'$ for some R' . Otherwise, $R \xrightarrow{da}^* R''$ for some R'' . In both cases $\{a, d\} \in C(R)$, which is a contradiction;
2. Suppose \mathcal{C}_1 and \mathcal{C}_2 are nested choices. Let $\mathcal{C}_2 \succ \mathcal{C}_1$ (the symmetric case is analogous). There are two subcases:
 - (a) either $\mathcal{C}_2 \succ \mathcal{B}_1$, in which case b (which is in \mathcal{B}_2) can never occur together with c or d (which are then in \mathcal{B}_1), thus contradicting $\{b, c\} \in C(R)$;
 - (b) or $\mathcal{C}_2 \succ \mathcal{B}_2$, in which case a (which is in \mathcal{B}_1) can never occur together with c or d (which are then in \mathcal{B}_2), thus contradicting $\{a, c\} \in C(R)$;
3. Suppose \mathcal{C}_1 and \mathcal{C}_2 are the same choice, namely $\{\mathcal{B}_1, \mathcal{B}_2\} = \{\mathcal{B}_3, \mathcal{B}_4\}$. Then, either $\mathcal{B}_1 = \mathcal{B}_3$, in which case b (which is in $\mathcal{B}_2 = \mathcal{B}_4$) cannot occur together with c (which is in $\mathcal{B}_1 = \mathcal{B}_3$), thus contradicting $\{b, c\} \in C(R)$; or $\mathcal{B}_1 = \mathcal{B}_4$, in which case a (which is in $\mathcal{B}_1 = \mathcal{B}_4$) cannot occur together with c (which is in $\mathcal{B}_2 = \mathcal{B}_3$), thus contradicting $\{a, c\} \in C(R)$;
4. Suppose \mathcal{C}_1 and \mathcal{C}_2 are mutually exclusive choices. Then, there must be some choice $\mathcal{C} = \{\mathcal{B}_5, \mathcal{B}_6\}$ such that $\mathcal{C}_1 \succ \mathcal{B}_5$ and $\mathcal{C}_2 \succ \mathcal{B}_6$. Consequently, a (which is in $\mathcal{B}_1 \succ \mathcal{B}_5$) cannot occur together with c (which is in $\mathcal{B}_3 \succ \mathcal{B}_6$), thus again contradicting $\{a, c\} \in C(R)$.

Since all cases are contradictory, we conclude that there does not exist any BP R such that $C(R) = C(S)$. \square

The following two lemmas state that the class of BPs is not included in the class of EBESs. Since EBESs are the most expressive class of static ESs in Figure 7 (it includes the class of PESs and BESs), this implies that no class of static ESs in Figure 7 includes that of BPs.

Lemma 29 ($BP \not\subseteq EBES$). *There exists a branching pomset R for which there does not exist an extended bundle event structure S such that $C(R) = C(S)$.*

Proof. Let R be the BP in Figure 8b. Assume, for the sake of contradiction, that there exists some EBES $S = \langle E, \sim, \succ \rangle$ with the same set of configurations, namely $C(S) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{d\}, \{a, c\}, \{a, d\}, \{b, c\}, \{b, d\}, \{a, c, d\}, \{b, c, d\}\}$. Since $\{a\}, \{b\}, \{c\}, \{d\} \in C(S)$, it follows that $\succ = \emptyset$. Furthermore, since $\{a, c, d\} \in C(S)$, it follows that $\neg(c \sim d) \vee \neg(d \sim c)$. However, it follows from these that $\{c, d\} \in C(S)$, which is a contradiction. \square

The following theorem follows directly from Theorems 28 and 29.

Theorem 30. *Branching pomsets are incomparable with the static event structures in Figure 7 (prime, bundle, and extended bundle).*

In contrast, for the special class of tree-like BPs, we can prove that it is strictly subsumed by the class of PESs.

Lemma 31. *For every tree-like branching pomset R there exists a prime event structure S such that $C(R) = C(S)$.*

Proof. Let $R = \langle E, \preceq, \lambda, \mathcal{B} \rangle$ be a BP. We construct the PES $S = \langle E, \#, \leq \rangle$, where $\leq = \preceq^*$, the reflexive and transitive closure of \preceq , and $\# = \{(e_1, e_2) \mid \exists \{\mathcal{B}_1, \mathcal{B}_2\} \succ \mathcal{B} : e_1 \succ \mathcal{B}_1 \wedge e_2 \succ \mathcal{B}_2\}$, i.e., the set of conflicts consists exactly of all pairs of events which are separated by some choice. Since R is tree-like, $e \succ \mathcal{B}' \wedge e \preceq e' \implies e' \succ \mathcal{B}'$, from which it follows that S satisfies conflict hereditariness. We proceed by showing that R and S have the same traces and thus the same configurations.

- Let $t = e_1 \dots e_n$ be a trace of R . Then $R \xrightarrow{t}^* R'$ for some R' . Since two mutually exclusive events can never occur in the same trace of R , t satisfies the PES consistency condition. Furthermore, for all $i \in \{1, \dots, n\}$, if there exists some $e \prec e_i$ then, since R is tree-like, $e \in \overline{t_{i-1}}$. In other words, t also satisfies the PES left-closure condition, and then t is also a trace of S .
- Let $t = e_1 \dots e_n$ be a trace of S . Then t must be consistent and left-closed. Let $i \in \{1, \dots, n\}$ and assume that $R \xrightarrow{t_{i-1}}^* R'$ for some R' (where $t_0 = \varepsilon$, in which case $R' = R$). Since t is left-closed, it follows that, for every e such that $e \leq e_i$, we have $e \in \overline{t_{i-1}}$. Consequently, e_i is minimal in R' . Furthermore, since t is consistent, e_i cannot be in conflict with any event in t_{i-1} and thus also does not belong to a different branch of any choice than the events in t_{i-1} . Since R is tree-like, there is no need to resolve choices for any reason other than firing events in them. It follows that $e_i \succ R'$, and then $R' \xrightarrow{e_i} R''$ for some R'' . Finally, it then follows by induction that $R \xrightarrow{t}^* \hat{R}$ for some \hat{R} and then t is a trace of R .

Since R and S have the same set of traces, it follows that $C(R) = C(S)$. \square

3.4. Comparing event structures and branching pomsets – Dynamic models

Our next two lemmas state that the class of BPs is not included in the classes of SESs and GESs. Combined with our result in the previous subsection that the class of PESs (subsumed by those of SESs and GESs) is not included in the class of BPs, we conclude that BPs and SESs/GESs are incomparable: the expressive power to only remove, or to only add, causalities dynamically is insufficient to cover the expressive power of BPs, and vice versa.

Lemma 32 (*BP $\not\subseteq$ SES*). *There exists a branching pomset R for which there does not exist a shrinking causality event structure S such that $C(R) = C(S)$.*

Proof. Let R be the BP in Figure 8b. Assume, for the sake of contradiction, that there exists some SES $S = \langle E, \#, \rightarrow, \triangleleft \rangle$ with the same set of configurations, namely $C(S) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{d\}, \{a, c\}, \{a, d\}, \{b, c\}, \{b, d\}, \{a, c, d\}, \{b, c, d\}\}$. Since $\{a\}, \{b\}, \{c\}, \{d\} \in C(S)$, it follows that $\rightarrow = \emptyset$. Furthermore, since $\{a, c, d\} \in C(S)$, it follows that $\neg(c \# d)$. However, it follows from these that $\{c, d\} \in C(S)$, which is a contradiction. \square

Lemma 33 ($BP \not\subseteq GES$). *There exists a branching pomset R for which there does not exist a growing causality event structure S such that $C(R) = C(S)$.*

Proof. Let R be the BP in Figure 8c. Assume, for the sake of contradiction, that there exists some GES $S = \langle E, \rightarrow, \blacktriangleright \rangle$ with the same set of configurations, namely $C(S) = \{\emptyset, \{a\}, \{b\}, \{a, c\}, \{b, c\}\}$. Since $\{c\} \notin C(S)$ and $\{a, c\} \in C(S)$, it follows that $a \rightarrow c$. However, since GESs cannot model disjunctive causality, it follows from $a \rightarrow c$ that $\{b, c\} \notin C(S)$, which is a contradiction. \square

The following theorem follows directly from Theorems 28, 32 and 33.

Theorem 34. *Branching pomsets are incomparable with two dynamic event structures in Figure 7 (growing and shrinking).*

In contrast, we conjecture that DCCESs (which combine the power of SESs and GESs) have more expressive power than BPs. The idea is that the power to remove dependencies can be used to encode disjunctive causality, while the power to add dependencies can be used to encode partial termination (as asymmetric conflicts).

Conjecture 35. For every branching pomset R there exists a dynamic causality event structure with counters S such that $R \simeq_t S$.

Finally, the general enabling relation of RCESs can essentially encode arbitrary transition graphs, including those induced by BPs. When equating traces of BPs to proving sequences for ESs, their definitions for configurations and (single action) transition relations (Theorems 22, 26 and 27) coincide. Then, occasionally substituting the word ‘trace’ for ‘proving sequence’ in the proof of Theorem 25 also proves the following theorem.

Theorem 36. *For every branching pomset R there exists an event structure for resolvable conflict S such that $R \simeq_t S$.*

4. Choreographies

We now turn to our study of applications of branching pomsets to choreographies. In this section we define a simple choreographic language. We then encode choreographic expressions into BPs and show the expressions’ operational semantics to be bisimilar to the encoding’s BP semantics.

$$c ::= \mathbf{1} \mid \mathbf{a} \rightarrow \mathbf{b} : \mathbf{x} \mid \boxed{\mathbf{ab} ? \mathbf{x}} \mid c ; c \mid c + c \mid c \parallel c \mid c^*$$

Figure 9: Syntax of choreographies, where \mathbf{a} and \mathbf{b} are participants ($\mathbf{a} \neq \mathbf{b}$) and \mathbf{x} is a message type.

Let $\mathcal{A} = \{\mathbf{a}, \mathbf{b}, \dots\}$ be the set of participants (or agents). Let $\mathcal{X} = \{\mathbf{x}, \mathbf{y}, \dots\}$ be the set of message types. From now on, let $\mathcal{L} = \bigcup_{\mathbf{a}, \mathbf{b} \in \mathcal{A}, \mathbf{x} \in \mathcal{X}} \{\mathbf{ab} ! \mathbf{x}, \mathbf{ab} ? \mathbf{x}\}$ be the set of labels (actions), ranged over by ℓ , where $\mathbf{ab} ! \mathbf{x}$ is a send action from \mathbf{a} to \mathbf{b} of a message of type \mathbf{x} , and $\mathbf{ab} ? \mathbf{x}$ is the corresponding receive action by \mathbf{b} . The *subject* of an action ℓ , written $\text{subj}(\ell)$, is its active agent: $\text{subj}(\mathbf{ab} ! \mathbf{x}) = \mathbf{a}$ and $\text{subj}(\mathbf{ab} ? \mathbf{x}) = \mathbf{b}$.

4.1. Choreography language definition

The syntax of our choreography language is formally defined in Figure 9. Its components are standard: ' $\mathbf{1}$ ' is the empty choreography; ' $\mathbf{a} \rightarrow \mathbf{b} : \mathbf{x}$ ' is the asynchronous communication from \mathbf{a} to \mathbf{b} of a message of type \mathbf{x} ; the boxed term ' $\mathbf{ab} ? \mathbf{x}$ ' represents a pending receive on \mathbf{b} from \mathbf{a} of a message of type \mathbf{x} (it is boxed in Figure 9 to indicate that it is only used internally to formalise behaviour but the box is not part of the syntax); ' $c_1 ; c_2$ ', ' $c_1 + c_2$ ' and ' $c_1 \parallel c_2$ ' are respectively the weak sequential composition, nondeterministic choice and parallel composition of choreographies c_1 and c_2 ; finally, ' c^* ' is the finite repetition (or, more informally, loop) of choreography c . The semantics for choice, parallel composition and loop are standard. We note that our sequential composition is weak. With standard sequential composition, when sequencing c_1 and c_2 , the choreography c_1 must fully terminate before proceeding to c_2 . With weak sequential composition, however, actions in c_2 can already be executed as long as they do not interfere with c_1 . For example, in $\mathbf{a} \rightarrow \mathbf{b} : \mathbf{x} ; \mathbf{c} \rightarrow \mathbf{d} : \mathbf{x}$ we can execute the action $\mathbf{cd} ! \mathbf{x}$ as it does not affect the participants of $\mathbf{a} \rightarrow \mathbf{b} : \mathbf{x}$: there is no dependency and thus no need to wait for $\mathbf{a} \rightarrow \mathbf{b} : \mathbf{x}$ to go first. On the other hand, in $\mathbf{a} \rightarrow \mathbf{b} : \mathbf{x} ; \mathbf{a} \rightarrow \mathbf{c} : \mathbf{x}$ the action $\mathbf{ac} ! \mathbf{x}$ cannot be executed first as its subject (\mathbf{a}) must first execute $\mathbf{ab} ! \mathbf{x}$. This is the common interpretation of sequential composition in the context of message sequence charts [29], multiparty session types [3] and choreographic programming [5].

The reduction rules of our choreographic language are formally defined in Figure 10a and its termination rules in Figure 10b. To formalise the reduction of weak sequential composition, we follow Rensink and Wehrheim [30], who define a notion of *partial termination*. Partial termination inspired our

$$\begin{array}{c}
\frac{}{\mathbf{a} \rightarrow \mathbf{b} : \mathbf{x} \xrightarrow{\text{ab!x}} \mathbf{ab?x}} [\rightarrow_1] \quad \frac{}{\mathbf{ab?x} \xrightarrow{\text{ab?x}} \mathbf{1}} [\rightarrow_2] \quad \frac{c_1 \xrightarrow{\ell} c'_1}{c_1 ; c_2 \xrightarrow{\ell} c'_1 ; c_2} [\rightarrow_3] \quad \frac{c_1 \xrightarrow{\ell} c'_1 \quad c_2 \xrightarrow{\ell} c'_2}{c_1 ; c_2 \xrightarrow{\ell} c'_1 ; c'_2} [\rightarrow_4] \\
\\
\frac{c_1 \xrightarrow{\ell} c'_1}{c_1 \parallel c_2 \xrightarrow{\ell} c'_1 \parallel c_2} [\rightarrow_5] \quad \frac{c_2 \xrightarrow{\ell} c'_2}{c_1 \parallel c_2 \xrightarrow{\ell} c_1 \parallel c'_2} [\rightarrow_6] \quad \frac{c_1 \xrightarrow{\ell} c'_1}{c_1 + c_2 \xrightarrow{\ell} c'_1} [\rightarrow_7] \quad \frac{c_2 \xrightarrow{\ell} c'_2}{c_1 + c_2 \xrightarrow{\ell} c'_2} [\rightarrow_8] \\
\\
\frac{c \xrightarrow{\ell} c'}{c^* \xrightarrow{\ell} c' ; c^*} [\rightarrow_9]
\end{array}$$

(a) Reduction rules.

$$\frac{}{\mathbf{1} \downarrow} [\downarrow_1] \quad \frac{}{c^* \downarrow} [\downarrow_2] \quad \frac{c_1 \downarrow \quad c_2 \downarrow \quad \dagger \in \{;, \parallel\}}{c_1 \dagger c_2 \downarrow} [\downarrow_3] \quad \frac{c_i \downarrow \quad i \in \{1, 2\}}{c_1 + c_2 \downarrow} [\downarrow_4]$$

(b) Termination rules.

$$\begin{array}{c}
\frac{}{\mathbf{1} \xrightarrow{\ell} \mathbf{1}} [\check{\rightarrow}_1] \quad \frac{c \xrightarrow{\ell} c}{c^* \xrightarrow{\ell} c^*} [\check{\rightarrow}_2] \quad \frac{c \not\xrightarrow{\ell} c}{c^* \xrightarrow{\ell} \mathbf{1}} [\check{\rightarrow}_3] \quad \frac{c_1 \xrightarrow{\ell} c'_1 \quad c_2 \xrightarrow{\ell} c'_2 \quad \dagger \in \{;, \parallel, +\}}{c_1 \dagger c_2 \xrightarrow{\ell} c'_1 \dagger c'_2} [\check{\rightarrow}_4] \\
\\
\frac{c_1 \xrightarrow{\ell} c'_1 \quad c_2 \not\xrightarrow{\ell} c'_2}{c_1 + c_2 \xrightarrow{\ell} c'_1} [\check{\rightarrow}_5] \quad \frac{c_1 \not\xrightarrow{\ell} c'_1 \quad c_2 \xrightarrow{\ell} c'_2}{c_1 + c_2 \xrightarrow{\ell} c'_2} [\check{\rightarrow}_6] \quad \frac{\text{subj}(\ell) \notin \{\mathbf{a}, \mathbf{b}\}}{\mathbf{a} \rightarrow \mathbf{b} : \mathbf{x} \xrightarrow{\ell} \mathbf{a} \rightarrow \mathbf{b} : \mathbf{x}} [\check{\rightarrow}_7] \quad \frac{\text{subj}(\ell) \neq \mathbf{b}}{\mathbf{ab?x} \xrightarrow{\ell} \mathbf{ab?x}} [\check{\rightarrow}_8]
\end{array}$$

(c) Partial termination rules.

Figure 10: Operational semantics of choreographies.

refinement and enabling rules for BPs, so both the concepts and notation are similar.

Partial termination. In a weak sequential composition $c_1 ; c_2$, an action ℓ in c_2 can be executed if c_1 can *partially terminate* for ℓ . Conceptually, a choreography c_1 can partially terminate for ℓ by discarding all branches of its behaviour which would conflict with it, i.e., in which the subject of ℓ occurs. This is written $c_1 \xrightarrow{\ell} c'_1$, where c'_1 is the remainder of c_1 after discarding all branches involving the subject of ℓ . For example, if $c_1 = \mathbf{a} \rightarrow \mathbf{b} : \mathbf{x} + \mathbf{a} \rightarrow \mathbf{c} : \mathbf{x}$ then $c_1 \xrightarrow{\text{cd!x}} \mathbf{a} \rightarrow \mathbf{b} : \mathbf{x}$, as this branch does not contain \mathbf{c} . An exception is when the subject of ℓ occurs in *every* branch of c_1 , in which case c_1 cannot partially

terminate for ℓ , i.e., $c_1 \not\check{\xrightarrow{\ell}}$. In the above example, $c_1 \not\check{\xrightarrow{\text{ad!x}}}$.

The rules for partial termination are deterministic and, like our enabling relation for BPs, only discard the absolutely necessary. In the example above, $c_1 \xrightarrow{\text{da!x}} c_1$ since the subject **d** does not occur in either branch: dropping one of the branches would be unnecessary and is thus not allowed. The rules for partial termination are defined in Figure 10c. We highlight the rules for the different operators:

- Sequential composition $c_1 ; c_2$ and parallel composition $c_1 \parallel c_2$ can partially terminate if both c_1 and c_2 can (rule $\check{\rightarrow}_4$).
- A choice $c_1 + c_2$ can partially terminate if at least one of its branches can. If both branches can partially terminate then both are kept (rule $\check{\rightarrow}_4$), otherwise only the partially terminated one is kept (rules $\check{\rightarrow}_5$ and $\check{\rightarrow}_6$).
- Following Rensink and Wehrheim, a loop c^* can partially terminate if its body (c) can partially terminate without discarding any branches, i.e., if $c \xrightarrow{\ell} c$. In that case also $c^* \xrightarrow{\ell} c^*$ (rule $\check{\rightarrow}_2$). Otherwise we allow c^* to be skipped entirely, represented as partial termination to **1**, i.e., $c^* \xrightarrow{\ell} \mathbf{1}$ (rule $\check{\rightarrow}_3$). This can happen either if c can partially terminate to c' but $c' \neq c$, or if c cannot partially terminate at all. We use $c \not\check{\xrightarrow{\ell}} c$ as a shorthand to cover both these cases. Skipping a loop is necessary, for example, in a protocol such as $(a \rightarrow b:x ; b \rightarrow a:x)^* ; a \rightarrow b:\text{done}$, in which Alice and Bob exchange an arbitrary number of messages **x** until Alice signals **done**. In this choreography, the loop has to partially terminate to **1** to eventually allow for the action **ab!done**.

Example 37. Let $c_1 \boxplus = (a \rightarrow b:x + a \rightarrow c:x) ; (d \rightarrow b:x + d \rightarrow e:x)$. Let $c_2 \boxplus = (a \rightarrow b:x + c \rightarrow b:x)^* \parallel (c \rightarrow a:x + c \rightarrow b:x)$.

- $c_1 \xrightarrow{\text{be!x}} a \rightarrow c:x ; d \rightarrow e:x$. The subject **b** of **be!x** occurs in one branch of each of both choices: $a \rightarrow b:x + a \rightarrow c:x \xrightarrow{\text{be!x}} a \rightarrow c:x$ (rule $\check{\rightarrow}_6$) and $d \rightarrow b:x + d \rightarrow e:x \xrightarrow{\text{be!x}} d \rightarrow e:x$ (rule $\check{\rightarrow}_6$), and then $c_1 \xrightarrow{\text{be!x}} a \rightarrow c:x ; d \rightarrow e:x$ (rule $\check{\rightarrow}_4$). While the recipient **e** also occurs in the second branch of the second choice, since it is not the actual subject it does not interfere with **be!x** because it is not its subject (rule $\check{\rightarrow}_7$).
- $c_1 \not\check{\xrightarrow{\text{ab!x}}}$. While the second choice can partially terminate without dropping any branches (rule $\check{\rightarrow}_4$), the first choice contains the subject **a** of

ab!x in both of its branches and none of rules $\check{\rightarrow}_4$, $\check{\rightarrow}_5$ and $\check{\rightarrow}_6$ apply. Since one of the choices cannot partially terminate, neither can their sequential composition: rule $\check{\rightarrow}_4$ does not apply.

- $c_2 \xrightarrow{\check{\rightarrow}_{\text{ad!x}}} \mathbf{1} \parallel c \rightarrow b:x$. The subject a of ad!x only occurs in one branch of the loop body, but, since rule $\check{\rightarrow}_2$ does not apply, the loop can only partially terminate to $\mathbf{1}$ through rule $\check{\rightarrow}_3$. On the right hand side of the parallel composition, a occurs only in the first branch and so rule $\check{\rightarrow}_6$ applies. The two sides are then combined through rule $\check{\rightarrow}_4$.
- $c_2 \not\xrightarrow{\check{\rightarrow}_{\text{cd!x}}}$. While the loop can again partially terminate to $\mathbf{1}$ through rule $\check{\rightarrow}_3$, the subject c of cd!x occurs in both branches of the right hand side of the parallel composition and none of rules $\check{\rightarrow}_4$, $\check{\rightarrow}_5$ and $\check{\rightarrow}_6$ apply. Since its right hand side cannot partially terminate, neither can it as a whole: rule $\check{\rightarrow}_4$ does not apply.

As already discovered by Rensink and Wehrheim [30], an unwanted consequence of these rules for partial termination is that unfolding iterations of loops no longer preserves behaviour. We would like c^* and $(c; c^*) + \mathbf{1}$ to behave the same, but this is not the case. For example, if $c \boxplus = \text{a} \rightarrow b:x + c \rightarrow d:x$, then $c^* \xrightarrow{\check{\rightarrow}_{\text{ab!x}}} \mathbf{1}$ (rule $\check{\rightarrow}_3$) but $(c; c^*) + \mathbf{1} \xrightarrow{\check{\rightarrow}_{\text{ab!x}}} (c \rightarrow d:x; \mathbf{1}) + \mathbf{1}$ (rules $\check{\rightarrow}_6$, $\check{\rightarrow}_3$ and $\check{\rightarrow}_4$). Then $c^*; c \xrightarrow{\check{\rightarrow}_{\text{ab!x}}} \mathbf{1}; \text{ab?x}$ by skipping the loop (rule \rightarrow_4); however, $((c; c^*) + \mathbf{1}); c$ has no way to match this as it can skip the loop but it can only partially terminate the already unfolded iteration c to $c \rightarrow d:x$ — it cannot discard it entirely. We borrow the solution that Rensink and Wehrheim offer, which is the concept *dependent guardedness*.

Dependent guardedness. A loop c^* is *dependently guarded* if, for all actions ℓ , the loop body c can only partially terminate for ℓ if it does not occur in c at all. In other words: any participant that occurs in some branch of c must also occur in every other branch of c . It then follows that c can either partially terminate for ℓ without having to discard any branches, or it cannot partially terminate at all. Formally: if $c \xrightarrow{\ell} c'$ then $c' = c$. A choreography \hat{c} is then dependently guarded if all of its loops are.

As a consequence, we avoid the problem above: if $c^* \xrightarrow{\ell} \mathbf{1}$ then $c \not\xrightarrow{\ell}$ and $c; c^* \not\xrightarrow{\ell}$ since rule $\check{\rightarrow}_4$ does not apply and, consequently, $(c; c^*) + \mathbf{1}$ is forced to partially terminate to the second branch of the choice (rule $\check{\rightarrow}_6$), which is $\mathbf{1}$. More precisely, let c^* be some dependently guarded expression. If $c \xrightarrow{\ell} c'$ for some ℓ, c' , then $c' = c$. It follows that $c^* \xrightarrow{\ell} c^*$ (rule $\check{\rightarrow}_2$) and

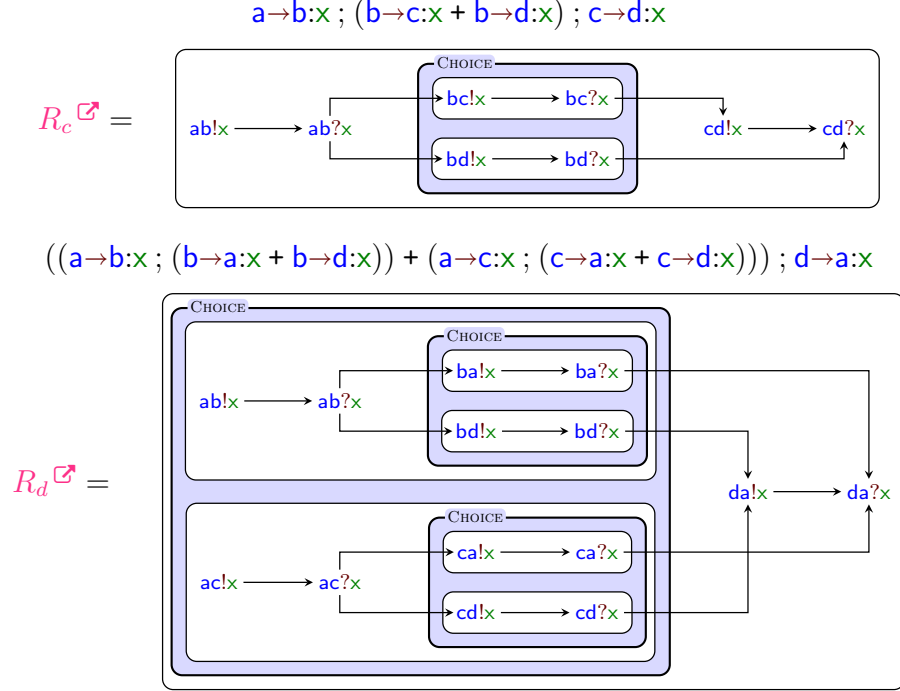


Figure 11: Two BPs with the corresponding choreographies.

$(c; c^*) + \mathbf{1} \xrightarrow{\ell} (c; c^*) + \mathbf{1}$ (rule $\xrightarrow{4}$). Similarly, if $c \not\xrightarrow{\ell}$ then $c^* \xrightarrow{\ell} \mathbf{1}$ (rule $\xrightarrow{3}$) and $(c; c^*) + \mathbf{1} \xrightarrow{\ell} \mathbf{1}$ (rule $\xrightarrow{6}$).

Example 38. Let $c_1 = a \rightarrow b:x + a \rightarrow c:x$. Let $c_2 = a \rightarrow b:x + b \rightarrow a:x$.

- $c_1^* \boxplus$ is not dependently guarded as $c_1 \xrightarrow{cd!x} a \rightarrow b:x \neq c_1$ (rule $\xrightarrow{5}$). However, c_1 itself is dependently guarded as it does not contain any loop.
- $c_2^* \boxplus$ is dependently guarded since both a and b occur in both branches of c_2 . However, $(c_2^*)^*$ is *not* dependently guarded, since $c_2^* \xrightarrow{ab!x} \mathbf{1}$ (rule $\xrightarrow{3}$).

4.2. BP encoding

Figure 11 shows two choreographies and corresponding BPs similar to those in Figure 4. Formally, the rules for the construction of a BP for a choreography c , written $\llbracket c \rrbracket$, are defined in Figure 12. Most rules are as expected. We highlight the rules for operators.

$$\begin{aligned}
\llbracket \mathbf{1} \rrbracket &= \langle \emptyset, \emptyset, \emptyset, \emptyset \rangle \\
\llbracket \mathbf{a} \rightarrow \mathbf{b} : \mathbf{x} \rrbracket &= \langle \{e_1, e_2\}, \{e_1 \preceq e_1, e_1 \preceq e_2, e_2 \preceq e_2\}, \{e_1 \mapsto \mathbf{ab}! \mathbf{x}, e_2 \mapsto \mathbf{ab}? \mathbf{x}\}, \{e_1, e_2\} \rangle \\
\llbracket \mathbf{ab}? \mathbf{x} \rrbracket &= \langle \{e\}, \{e \preceq e\}, \{e \mapsto \mathbf{ab}? \mathbf{x}\}, \{e\} \rangle \\
\llbracket c_1 \dagger c_2 \rrbracket &= \llbracket c_1 \rrbracket \dagger \llbracket c_2 \rrbracket \text{ for } \dagger \in \{;, +, \parallel\} \\
\llbracket c^* \rrbracket &= \llbracket (c ; c^*) + \mathbf{1} \rrbracket
\end{aligned}$$

In the following, let $R_i = \langle E_i, \preceq_i, \lambda_i, \mathcal{B}_i \rangle$ for $i \in \{1, 2\}$ and let $E_i^{\mathbf{a}}$ be the subset of events in E_i with subject \mathbf{a} .

$$\begin{aligned}
R_1 \parallel R_2 &= \langle E_1 \cup E_2, \preceq_1 \cup \preceq_2, \lambda_1 \cup \lambda_2, \mathcal{B}_1 \cup \mathcal{B}_2 \rangle \\
R_1 ; R_2 &= \langle E_1 \cup E_2, \preceq_1 \cup \preceq_2 \cup \bigcup_{\mathbf{a} \in \mathcal{A}} E_1^{\mathbf{a}} \times E_2^{\mathbf{a}}, \lambda_1 \cup \lambda_2, \mathcal{B}_1 \cup \mathcal{B}_2 \rangle \\
R_1 + R_2 &= \langle E_1 \cup E_2, \preceq_1 \cup \preceq_2, \lambda_1 \cup \lambda_2, \{\{\mathcal{B}_1, \mathcal{B}_2\}\} \rangle
\end{aligned}$$

Figure 12: BP interpretation of choreographies.

- The rule for parallel composition ($\llbracket c_1 \parallel c_2 \rrbracket$) takes the pairwise union of all components.
- The rule for sequential composition ($\llbracket c_1 ; c_2 \rrbracket$) also adds dependencies to ensure that, for every \mathbf{a} , all events with subject \mathbf{a} in $\llbracket c_1 \rrbracket$ (denoted $E_1^{\mathbf{a}}$) must precede all events with subject \mathbf{a} in $\llbracket c_2 \rrbracket$ ($E_2^{\mathbf{a}}$). This matches the reduction rule for weak sequential composition of choreographies (Figure 10a), as events in $\llbracket c_2 \rrbracket$ are only required to wait for events in $\llbracket c_1 \rrbracket$ whose subject is the same.
- The rule for choice ($\llbracket c_1 + c_2 \rrbracket$) adds a single top-level choice in the branching structure to choose between the BPs for c_1 and c_2 .
- The rule for loops ($\llbracket c^* \rrbracket$) encodes a loop as a choice between terminating ($\mathbf{1}$) and unfolding one iteration of the loop ($c ; c^*$). This results in a BP of infinite size. We note that our theoretical results still hold even on infinite BPs, but that any analysis of an infinite BP will have to be symbolic. However, since the focus of this paper is on supporting choices, we do not discuss this further and leave symbolic analyses for loops for future work.

Example 39. As an example, we construct part of the BP in Figure 2: $\llbracket (c \rightarrow a:r; (a \rightarrow c:y + a \rightarrow c:n)) \parallel (c \rightarrow b:r; (b \rightarrow c:y + b \rightarrow c:n)) + 1 \rrbracket$ (thus omitting $c \rightarrow a:t$ and $c \rightarrow b:t$). Let $\llbracket c \rightarrow a:r \rrbracket = \langle \{e_1, e_2\}, \preceq_1, \lambda_1, \mathcal{B}_1 \rangle$, $\llbracket a \rightarrow c:y \rrbracket = \dots$, $\llbracket a \rightarrow c:n \rrbracket = \dots$, $\llbracket c \rightarrow b:r \rrbracket = \dots$, $\llbracket b \rightarrow c:y \rrbracket = \dots$ and $\llbracket b \rightarrow c:n \rrbracket = \langle \{e_{11}, e_{12}\}, \preceq_6, \lambda_6, \mathcal{B}_6 \rangle$ as in Figure 12. First $\llbracket a \rightarrow c:y + a \rightarrow c:n \rrbracket = \langle \{e_3, \dots, e_6\}, \preceq_2 \cup \preceq_3, \lambda_2 \cup \lambda_3, \{\{\mathcal{B}_2, \mathcal{B}_3\}\} \rangle$; this is the pairwise union of the first three components, with the branching structure adding a choice between the two branches. Then $\llbracket c \rightarrow a:r; (a \rightarrow c:y + a \rightarrow c:n) \rrbracket = \langle \{e_1, \dots, e_6\}, \preceq_1 \cup \preceq_2 \cup \preceq_3 \cup \{e_1 \preceq e_4, e_1 \preceq e_6, e_2 \preceq e_3, e_2 \preceq e_5\}, \lambda_1 \cup \lambda_2 \cup \lambda_3, \mathcal{B}_1 \cup \{\{\mathcal{B}_2, \mathcal{B}_3\}\} \rangle$; again, this is the pairwise union of all components, with the addition of four dependencies: $e_2 \preceq e_3$ and $e_2 \preceq e_5$ represent the arrows in Figure 2 from $ca!r$ to respectively $ac!y$ and $ac!n$ as they both have subject a , while $e_1 \preceq e_4$ and $e_1 \preceq e_6$ adds direct dependencies between $ca!r$ and respectively $ac?y$ and $ac?n$ as they both have subject c . The parallel branch involving Bob is analogous, and their parallel composition simply consists of the pairwise union of their components. Finally, adding the choice with 1 retains the first three components and yields the branching structure $\{\{\mathcal{B}_p, \emptyset\}\}$, where \mathcal{B}_p is the branching structure yielded by the parallel composition.

Remark (expressiveness). There is no way to obtain the precise BP from Figure 2, as parallel composition of the BP constructed in Theorem 39 with $\llbracket c \rightarrow a:t \parallel c \rightarrow b:t \rrbracket$ yields too few dependencies and sequential composition yields too many: amongst others, sequential composition would also add arrows from $ac?y$ and $ac?n$ to $cb!t$, as they have the same subject c . The behaviour can be expressed by duplicating events, resulting in the choreography $c_{\text{snd}}^{\text{lenient}}$ in Section 1. Expressing it without duplication would require more sophisticated compositional operators than the ones in our choreographic language. As a second example, there is also no way to obtain the BP from Figure 13. In this BP, Alice (a) and Bob (b) both send the other a vote (v), but they must send their own vote before receiving the other's. From our choreographic language we can obtain BPs for $a \rightarrow b:v$ and $b \rightarrow a:v$, but we have no compositional operator to compose them in the desired way: parallel composition adds no dependencies at all, and sequential composition will also enforce an ordering on the send and receive events.

4.3. Equivalence of BP encoding

For any given choreography c we can now derive two labelled transition systems: one from the operational semantics in Figure 10 over c , and one

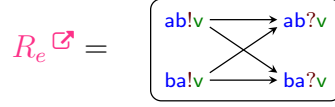


Figure 13: A BP representing a distributed vote with two voters.

from the pomset semantics in Figure 3 over the BP $\llbracket c \rrbracket$ produced by the rules in Figure 12. In the remainder of this section we show that the two transition systems are bisimilar.

Two systems are language equivalent (or trace equivalent) if their languages are the same, i.e., if they accept the same set of words (or traces), regardless of the way these words are obtained. On the other hand, two systems are bisimilar if their internal branching behaviour is also the same. This is a stronger notion of equivalence than language equivalence: if two systems are bisimilar then they are also language equivalent, but the inverse is not necessarily true.

Example 40.

- $a \rightarrow b:x; (b \rightarrow a:x + b \rightarrow a:y)$ is language equivalent but not bisimilar to $(a \rightarrow b:x; b \rightarrow a:x) + (a \rightarrow b:x; b \rightarrow a:y)$. In the former the choice between $b \rightarrow a:x$ and $b \rightarrow a:y$ is made only after $a \rightarrow b:x$, while in the latter the choice is made up front. As a result, it is possible in the latter system to fire $ab!x; ab?x$ and then end up in a state where $ba!x$ cannot be fired because the branch with $b \rightarrow a:y$ was chosen — or the other way around; in the former system it is always possible to fire both $ba!x$ and $ba!y$.
- $a \rightarrow b:x$ is bisimilar to $a \rightarrow b:x + a \rightarrow b:x$. While the latter contains a choice, the two systems cannot be distinguished by their behaviour. In both cases, the only allowed action is $ab!x$ and then $ab?x$.

Formally, two transition systems A_1, A_2 are bisimilar, written $A_1 \sim A_2$, if there exists a bisimulation relation \mathcal{R} between the states of A_1 and A_2 which relates their initial states [31]. The relation \mathcal{R} is a bisimulation relation if, for every pair of states $\langle p, q \rangle \in \mathcal{R}$:

- If $p \xrightarrow{\ell} p'$ then $q \xrightarrow{\ell} q'$ and $\langle p', q' \rangle \in \mathcal{R}$ for some q' , and vice-versa.
- $p \downarrow$ iff $q \downarrow$.

In other words: if one of the two can perform a step, then the other can perform a matching step such that the resulting states are again in the bisimulation relation.

This is also the approach we follow when proving that $c \sim \llbracket c \rrbracket$ for all (dependently guarded) choreographies c : we define a relation $\mathcal{R} = \{\langle c, \llbracket c \rrbracket \rangle \mid c \text{ is a dependently guarded choreography}\}$ relating all dependently guarded choreographies with their interpretation as BP by the rules in Figure 12. We then show that:

- If $c \xrightarrow{\ell} c'$ then $\llbracket c \rrbracket \xrightarrow{\ell} \llbracket c' \rrbracket$ (Theorem 42).
- If $\llbracket c \rrbracket \xrightarrow{\ell} R'$ then $c \xrightarrow{\ell} c'$ such that $R' = \llbracket c' \rrbracket$ (Theorem 43).
- If $c \downarrow$ then $\llbracket c \rrbracket \downarrow$ (Theorem 44).
- If $\llbracket c \rrbracket \downarrow$ then $c \downarrow$ (Theorem 45).

Together these lemmas prove that $c \sim \llbracket c \rrbracket$ for all dependently guarded c (Theorem 46). Most of the proofs are straightforward by structural induction on c . Of particular interest, however, are the two reduction lemmas in the case of weak sequential composition, i.e., if $c_1; c_2 \xrightarrow{\ell} c'_1; c'_2$ in Theorem 42 and if $\llbracket c_1; c_2 \rrbracket \xrightarrow{e} R'$ where e is an event in $\llbracket c_2 \rrbracket$ in Theorem 43. To prove these specific cases we need to show a correspondence between partial termination and enabling events. We do this with Theorem 41, in which we show two directions simultaneously. If the choreography c_1 can partially terminate for an action ℓ in c_2 then the BP $\llbracket c_1; c_2 \rrbracket$ can enable the corresponding event. Conversely, if $\llbracket c_1; c_2 \rrbracket$ can enable some event in $\llbracket c_2 \rrbracket$ then the choreography c_1 can partially terminate for its label. When proving these cases in Theorems 42 and 43, we then only have to show that the preconditions of Theorem 41 hold.

In the following, a number of technical lemmas and most of the proofs are omitted in favour of informal proof sketches or highlights. The omitted proofs and technical lemmas for this section can be found in the technical report of our ICE 2022 paper [32].

Lemma 41. *Let c_1 and c_2 be dependently guarded choreographies. Let $c_2 \xrightarrow{\ell} c'_2$ and $\llbracket c_2 \rrbracket \xrightarrow{\check{e}} R'_2$ such that $\lambda(e) = \ell$ and $\llbracket c'_2 \rrbracket = R'_2 - e$.*

- (a) *If $c_1 \xrightarrow{\check{e}} c'_1$ then $\llbracket c_1; c_2 \rrbracket \xrightarrow{\check{e}} \llbracket c'_1 \rrbracket; R'_2$.*
- (b) *If $\llbracket c_1; c_2 \rrbracket \xrightarrow{\check{e}} R'_1; R'_2$ then $c_1 \xrightarrow{\check{\lambda}(e)} c'_1$ and $\llbracket c'_1 \rrbracket = R'_1$.*

Proof sketch. This proof is by structural induction on c_1 . Although the details require careful consideration, it is conceptually straightforward: every case in (a) consists of showing that e is minimal and active in $\llbracket c'_1 \rrbracket ; R'_2$ and that $\llbracket c'_1 \rrbracket ; R'_2$ is the first refinement for which this is true, and then applying the second rule in Figure 3b; every case in (b) consists of showing that $\llbracket c_3 ; c_2 \rrbracket \xrightarrow{e} \llbracket c'_3 \rrbracket ; R'_2$ for some subexpression c_3 of c_1 and similarly for c_4 (e.g., when $c_1 = c_3 + c_4$), then applying the induction hypothesis (b) to obtain $c_3 \xrightarrow{\ell} c'_3$ and $c_4 \xrightarrow{\ell} c'_4$, and finally applying the partial termination rules in Figure 10c. \square

Lemma 42. *Let c be a dependently guarded choreography. If $c \xrightarrow{\ell} c'$ then $\llbracket c \rrbracket \xrightarrow{\ell} \llbracket c' \rrbracket$.*

Proof sketch. This proof is by structural induction on c . We note that, if $c = c_1 ; c_2$ and $c' = c'_1 ; c'_2$, i.e., when partial termination is applied, then the premises of Theorem 41 hold by the induction hypothesis and the result swiftly follows. All other cases are straightforward. \square

Lemma 43. *Let c be a dependently guarded choreography. If $\llbracket c \rrbracket \xrightarrow{\ell} R'$ for some R' then $c \xrightarrow{\ell} c'$ such that $R' = \llbracket c' \rrbracket$.*

Proof sketch. This proof is by structural induction on c . We highlight two cases:

- If $c = c_1^*$ then we use a technical lemma to show that $R' = R'_1 ; \llbracket c_1^* \rrbracket$ such that $\llbracket c_1 \rrbracket \xrightarrow{\ell} R'_1$. It then follows from the induction hypothesis that $c_1 \xrightarrow{\ell} c'_1$ such that $\llbracket c'_1 \rrbracket = R'_1$. The remainder is straightforward.
- If $c = c_1 ; c_2$ then $\llbracket c \rrbracket = \llbracket c_1 \rrbracket ; \llbracket c_2 \rrbracket$. If e is an event in $\llbracket c_2 \rrbracket$ then we proceed to show that $\llbracket c_2 \rrbracket \xrightarrow{\ell} R'_2$, at which point we can apply the induction hypothesis. We have then satisfied the premises of Theorem 41. The remainder is straightforward.

All other cases are straightforward. \square

Lemma 44. *Let c be a dependently guarded choreography. If $c \downarrow$ then $\llbracket c \rrbracket \downarrow$.*

Proof sketch. This proof is by structural induction on c . All cases are straightforward. \square

Lemma 45. *Let c be a dependently guarded choreography. If $\llbracket c \rrbracket \downarrow$ then $c \downarrow$.*

Proof sketch. This proof is by structural induction on c . All cases are straightforward. \square

Theorem 46. *Let c be a dependently guarded choreography. Then $c \sim \llbracket c \rrbracket$.*

Proof. Recall the relation $\mathcal{R} = \{\langle c, \llbracket c \rrbracket \rangle \mid c \text{ is a dependently guarded choreography}\}$. Let $\langle c, R \rangle \in \mathcal{R}$.

- If $c \xrightarrow{\ell} c'$ then $R \xrightarrow{\ell} R'$ and $\langle c', R' \rangle \in \mathcal{R}$ (Theorem 42).
- If $R \xrightarrow{\ell} R'$ then $c \xrightarrow{\ell} c'$ and $\langle c', R' \rangle \in \mathcal{R}$ (Theorem 43).
- If $c \downarrow$ then $R \downarrow$ (Theorem 44).
- If $R \downarrow$ then $c \downarrow$ (Theorem 45).

Then \mathcal{R} is a bisimulation relation and $c \sim \llbracket c \rrbracket$ ([31]). \square

5. Realisability

In this section we formally define the realisability property for BPs representing choreographies. For our analysis, we draw inspiration from multi-party session types (MPST) [3]. Through its syntax and projection operator, MPST defines a number of well-formedness conditions on global types which ensure realisability. We define similar well-formedness conditions on BPs and prove that they ensure realisability as well. These conditions are sufficient but not necessary, i.e., a protocol may be realisable without being well-formed. We discuss some possible relaxations of the conditions at the end of the paper.

As in the previous section, we omit a number of technical lemmas and most of the proofs in favour of informal proof sketches or highlights. The omitted proofs and lemmas for this section can be found in the technical report of our FACS 2022 paper [33].

5.1. Realisability of BPs

We model distributed implementations as compositions of a collection of local BPs \vec{R} and ordered buffers (FIFO queues) \vec{b} containing the messages in transit (sent but not yet received) between directed pairs of agents (or channels), similar to communicating finite-state machines [34]. The local BPs only contain actions for a single agent; there should be one local BP for each agent and one buffer for each channel.

Composition is formally defined below. We use three auxiliary functions: $add(ab!x, \vec{b})$ returns \vec{b} with x added to b_{ab} , $remove(ab!x, \vec{b})$ returns \vec{b} with x removed from b_{ab} and $has(ab!x, \vec{b})$ returns whether x is pending in b_{ab} . Since we consider ordered buffers, add appends message types to the end of the corresponding queue, $remove$ removes message types from the front, and has only checks whether the first message matches.

We note that our termination condition does not require the buffers to be empty. In practice asynchronous communication channels will typically have some latency, and requiring empty buffers would require processes (the local BPs) to be aware of messages in transit. Instead, in our model the presence or absence of orphan messages (messages unreceived on termination) is a separate property from realisability, to be verified in isolation. It does, however, follow from our well-formedness conditions in [Section 5.2](#) that a well-formed and realisable protocol is also free of orphan messages.

Definition 47 (Composition). Let \vec{R} be an agent-indexed vector of local BPs. Let \vec{b} be a channel-indexed vector of ordered buffers. Their composition is the tuple $\langle \vec{R}, \vec{b} \rangle$, whose semantics is defined as the labeled transition system defined by the rules below.

$$\begin{array}{c}
\frac{R_a \xrightarrow{ab!x} R'_a \quad \vec{b}' = add(ab!x, \vec{b})}{\langle \vec{R}, \vec{b} \rangle \xrightarrow{ab!x} \langle \vec{R}[R'_a/R_a], \vec{b}' \rangle} [\text{SEND}] \quad \frac{R_b \xrightarrow{ab?x} R'_b \quad \vec{b}' = remove(ab!x, \vec{b})}{\langle \vec{R}, \vec{b} \rangle \xrightarrow{ab?x} \langle \vec{R}[R'_b/R_b], \vec{b}' \rangle} [\text{RECEIVE}] \\
\\
\frac{\forall a : R_a \downarrow}{\langle \vec{R}, \vec{b} \rangle \downarrow} [\text{TERMINATE}]
\end{array}$$

A protocol is *realisable* if there exists a faithful distributed implementation of it, i.e., one defining the same behaviour. We formally define realisability below. We note that it is typically defined in terms of language (trace) equivalence [\[13\]](#). However, as the exact branching of choices plays an important part in BPs, we use a more strict notion of equivalence and require the global BP and the composition to be bisimilar [\[31\]](#), as in [Section 4](#). We note that our well-formedness conditions enforce a deterministic setting, in which bisimilarity agrees with language equivalence. We then choose to prove bisimilarity rather than language equivalence because the proofs are typically more straightforward.

Recall from [Section 4](#) that two BPs R_1, R_2 are bisimilar, written $R_1 \sim R_2$, if, for every reduction $R_1 \xrightarrow{\ell} R'_1$ there exists a reduction $R_2 \xrightarrow{\ell} R'_2$ such that R'_1 and R'_2 are again bisimilar, and vice-versa. Additionally, we require that two bisimilar BPs R_1, R_2 can terminate iff the other can do so as well.

Definition 48 (Realisability). Let R be a BP. The protocol it represents is realisable if there exists a composition $\langle \vec{R}, \vec{b} \rangle$ such that b_{ab} is empty for all a, b and $R \sim \langle \vec{R}, \vec{b} \rangle$.

Example 49. Consider the BPs in [Figure 14](#):

- R_f is unrealisable. Alice and Bob both have to send a **yes** or a **no** to the other but the two messages must be the same. It is impossible without further synchronisation or communication to prevent a scenario in which one will send a different message than the other.
- R_g is realisable. Alice first sends an **int** and then a **bool** to Bob. After receiving the **int**, Bob returns either a **yes** or a **no**.
- R_h is unrealisable. Alice sends an **int** and a **bool** to Bob, but while they agree that Alice first sends the **int** and then the **bool**, the order in which Bob receives the message is unspecified. As we assume ordered buffers, Bob will first receive the **int**, but the global BP allows an execution in which Bob first receives the **bool**.
- R_i is realisable. Alice sends a **yes** or a **no** to Bob, followed by an **int**.

The second stage of the review protocol in [Figure 2](#) is realisable as well. Each choice is resolved by a single agent, and there is no way for the other (relevant) agents to confuse the different branches, nor is there an ordering issue such as for R_h .

We note that it is easy to go from a global BP R to a local BP for some agent a by *projecting* it on a , written $R|_a$. We will use projections in our well-formedness conditions and realisability proof, and we formally define them below. The projection $R|_a$ restricts R to the events whose subject is a , and restricts \preceq and λ accordingly. The branching structure is pruned by removing all discarded events (leaves), but no inner nodes of the tree are removed, even if they are left without any children. This is done to safeguard the symmetry with the branching structure of R to ease our proofs.

Definition 50 (Projection). $\langle E, \preceq, \lambda, \mathcal{B} \rangle|_a = \langle E_a, \preceq_a, \lambda_a, \mathcal{B}_a \rangle$ where:

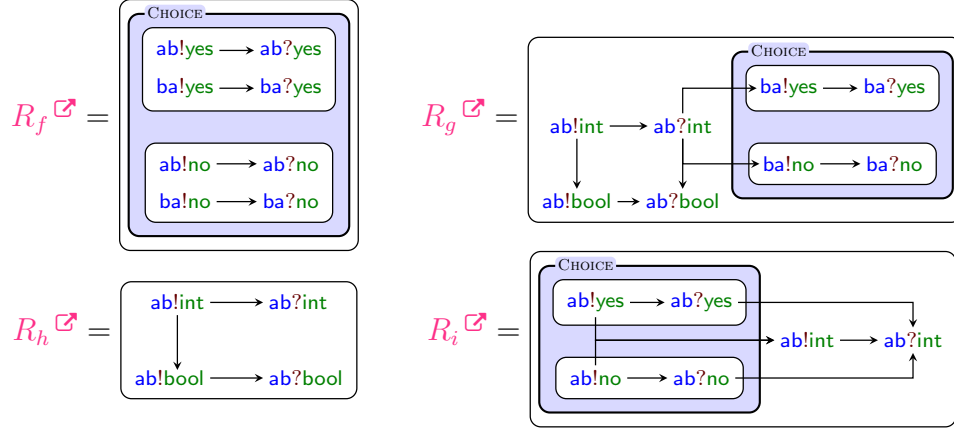


Figure 14: A collection of realisable and unrealisable BPs.

- $E_a = \{e \in E \mid \text{subj}(e) = a\}$
- $\preceq_a = \preceq \cap (E_a \times E_a)$
- $\lambda_a = \lambda \cap (E_a \times \mathcal{L})$
- $\mathcal{B}_a = \mathcal{B}|_a$ as defined below.

$$\begin{aligned}
 e|_a &= e \text{ if } e \in E_a \\
 \{\mathcal{C}_1, \dots, \mathcal{C}_n\}|_a &= \{\mathcal{C}_i|_a \mid 1 \leq i \leq n \wedge \mathcal{C}_i|_a \text{ is defined}\} \\
 \{\mathcal{B}_1, \mathcal{B}_2\}|_a &= \{\mathcal{B}_1|_a, \mathcal{B}_2|_a\}
 \end{aligned}$$

As an example, Figure 15 shows the projections of the BP for the second stage of the review protocol in Figure 2 on Carol and Alice. The projection on Bob is analogous to that on Alice. The events with different subjects are removed, as are dependencies involving them. We note that, as the graphical representation of a BP shows the transitive reduction of the causality relation and not the full relation, it is unclear from just Figure 2 whether, for example, the projection on Carol should contain dependencies between $ca!r$ and respectively $ac?y$ and $ac?n$. This is unambiguous in the formal textual definition of this example, which we omitted but which also relates these events.

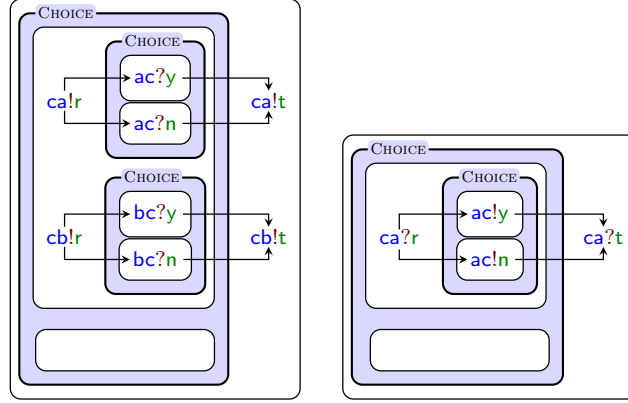


Figure 15: The projection of the BP in Figure 2 on c (left) and a (right).

Finally, we prove that R and its projections can mirror each other's refinements. Both proofs are straightforward by induction on the structure of the premise's derivation tree.

Lemma 51. *If $R \sqsupseteq R'$ then $R|_a \sqsupseteq R'|_a$.*

Lemma 52. *If $R|_a \sqsupseteq R'_a$ then $R \sqsupseteq R'$ for some R' such that $R'_a = R'|_a$.*

5.2. Well-formedness

We define four well-formedness conditions (Theorem 58): to be well-formed, a BP must be well-branched, well-channeled, tree-like and choreographic. Well-branchedness, well-channeledness and tree-likeness are inspired by MPST [3] and ensure some safety properties. Choreographicness ensures that the BP represents some sort of meaningful protocol.

- **Well-branched** (Theorem 54): every choice is made only on the label of a send-receive pair, i.e., the first events in every branch must be a send and receive between some agents a and b , with the message type being different in every branch. Additionally, the projection on every agent uninvolved in the choice must be the same in every branch. Then a and b are both aware of the chosen branch and all other agents are unaffected by the choice.

Although the BP model only contains binary choices, an n -ary choice \mathcal{C} can be encoded as a nested binary one, where the n children of \mathcal{C} become the leaves of a binary tree. We call such a leaf \mathcal{B} an *option*.

of \mathcal{C} , written $\mathcal{B} \text{opt } \mathcal{C}$, which is formally defined below. This allows us to properly interpret \mathcal{C} as an n -ary choice again and reason about it accordingly.

- **Well-channelled** (Theorem 55): pairs of sends and pairs of receives on the same channel that can occur in the same execution should be ordered, and the pairs of sends should have the same order as the pairs of their corresponding receives. A BP which is not well-channelled could, for example, yield a trace $\text{ab!x} ; \text{ab!y} ; \text{ab?y} ; \text{ab?x}$, which cannot be reproduced by a composition using ordered buffers.
- **Tree-like** (Theorem 56): events inside of choices can only affect future events in the same branch. Graphically speaking, arrows can only enter boxes, not leave them. As a consequence, the causality relation \preceq follows the branching structure \mathcal{B} and has a tree-like shape — hence the name.
- **Choreographic** (Theorem 57): the BP represents a choreography of some sort, i.e., a communication protocol in which the send and receive events are properly paired and all dependencies can be logically derived. Specifically, all dependencies are between send-receive pairs or between same-subject events, or they can be transitively derived from those. Additionally, there is the following correspondence between the send and receive events: every send can be matched to exactly one corresponding receive, and every non-top-level receive has some corresponding send at the same level of the branching structure \mathcal{B} . This definition is similar to the definition of well-formedness by Guanciale and Tuosto [13].

Definition 53 (Option). Let $\mathcal{C} \succ R.\mathcal{B}$. \mathcal{B} is an *option* of \mathcal{C} , written $\mathcal{B} \text{opt } \mathcal{C}$, if $\mathcal{B} \in \{\mathcal{B}^\dagger \mid \mathcal{B}^\dagger \text{opt}^\dagger \mathcal{C} \wedge \nexists \mathcal{B}^\ddagger : (\mathcal{B}^\dagger \text{opt}^\dagger \mathcal{C} \wedge \mathcal{B}^\ddagger \succ \mathcal{B}^\ddagger)\}$, where opt^\dagger is defined as follows:

$$\frac{\mathcal{B} \in \mathcal{C}}{\mathcal{B} \text{opt}^\dagger \mathcal{C}} \quad \frac{\mathcal{B} \in \mathcal{C}' \quad \{\mathcal{C}'\} \text{opt}^\dagger \mathcal{C}}{\mathcal{B} \text{opt}^\dagger \mathcal{C}}$$

Definition 54 (Well-branched). A BP R is *well-branched* if, for every $\mathcal{C} \succ R.\mathcal{B}$ there exist participants \mathbf{a}, \mathbf{b} such that for every $\mathcal{B}_i \neq \mathcal{B}_j \text{opt } \mathcal{C}$ there exist events $e_{i1}, e_{i2} \in \mathcal{B}_i$ and $e_{j1}, e_{j2} \in \mathcal{B}_j$ such that:

- $\lambda(e_{i1}) = \text{ab!x}$, $\lambda(e_{i2}) = \text{ab?x}$, $\lambda(e_{j1}) = \text{ab!y}$ and $\lambda(e_{j2}) = \text{ab?y}$ for some $\mathbf{x} \neq \mathbf{y}$;

- $e_{i1} \preceq e_i$ for all $e_i \succeq \mathcal{B}_i$ and $e_{j1} \preceq e_j$ for all $e_j \succeq \mathcal{B}_j$;
- $e_{i2} \preceq e_i$ for all $e_i \succeq \mathcal{B}_i$ for which $\text{subj}(e_i) = \mathbf{b}$ and $e_{j2} \preceq e_j$ for all $e_j \succeq \mathcal{B}_j$ for which $\text{subj}(e_j) = \mathbf{b}$; and
- $R|_{\mathcal{B}_i}|_{\mathbf{c}} = R|_{\mathcal{B}_j}|_{\mathbf{c}}$ for all $\mathbf{c} \neq \mathbf{a}, \mathbf{b}^{10}$.

Definition 55 (Well-channeled). A BP R is well-channeled if, for all events $e_1, e_2, e_3, e_4 \in R.E$:

- If e_1 and e_2 are either both sends or both receive events, and if they share the same channel, then they are either causally ordered or mutually exclusive.
- If e_1, e_3 and e_2, e_4 are two pairs of matching send-receive events sharing the same channel, and if there exists no $e_5 \in R.E$ such that $e_1 \prec e_5 \prec e_3$ or $e_2 \prec e_5 \prec e_4$, then $e_1 \preceq e_2 \implies e_3 \preceq e_4$.

Definition 56 (Tree-like). A BP R is *tree-like* if:

$$\forall \mathcal{C} = \{\mathcal{B}_1, \mathcal{B}_2\} \succ R.\mathcal{B} : (e_1 \preceq e_2 \wedge e_1 \succeq \mathcal{B}_i) \implies e_2 \succeq \mathcal{B}_i, \text{ where } i \in \{1, 2\}.$$

Definition 57 (Choreographic). A BP R is *choreographic* if, for every $e \in R.E$:

- If there exists $e' \in R.E$ such that $e' \prec e$ then there exists some event e'' (not necessarily distinct from e') such that $e' \preceq e'' \prec e$ and either $\text{subj}(\lambda(e'')) = \text{subj}(\lambda(e))$ or $[\lambda(e'') = \mathbf{ab}!\mathbf{x}]$ and $\lambda(e) = \mathbf{ab}?\mathbf{x}$ for some $\mathbf{a}, \mathbf{b}, \mathbf{x}$.
- If $\lambda(e) = \mathbf{ab}?\mathbf{x}$ and $e \in \mathcal{B}$ for some $\mathcal{B} \succ R.\mathcal{B}$ then there exists some e' such that $e' \in \mathcal{B}$ and $\lambda(e') = \mathbf{ab}!\mathbf{x}$ and $e' \prec e$.
- If $\lambda(e) = \mathbf{ab}!\mathbf{x}$ then there exists exactly one e' such that $e \preceq e'$ and that $\lambda(e') = \mathbf{ab}?\mathbf{x}$ and that $\forall e^\dagger : (\lambda(e^\dagger) = \mathbf{ab}!\mathbf{x} \wedge e^\dagger \preceq e') \implies e^\dagger \preceq e$.

Definition 58 (Well-formed). A BP R is *well-formed* if it is well-branched, well-channeled, tree-like and choreographic.

¹⁰Technically $R|_{\mathcal{B}_i}|_{\mathbf{c}}$ and $R|_{\mathcal{B}_j}|_{\mathbf{c}}$ have different events and should thus be isomorphic rather than precisely equal. We choose to write it as an equality to not unnecessarily complicate the definition and proofs.

Example 59. Recall the BPs in Figure 14:

- R_f is not well-formed since it is not well-branched: for example, the branches of the choice have multiple minimal events. It is indeed unrealisable.
- R_g is both well-formed and realisable.
- R_h is not well-formed since it is not well-channeled: the two receive events are on the same channel but are unordered. It is indeed unrealisable.
- R_i is not well-formed since it is not tree-like: there are arrows from events inside branches of a choice to $ab!int$ and $ab?int$, even though the latter are not part of the same branch. It is, however, realisable: by duplicating the events $ab!int$ and $ab?int$ and moving one copy inside each branch, we obtain an equivalent BP which is well-formed. This illustrates that, while we later prove that our well-formedness conditions are sufficient, they are not necessary.

The BP for the second stage of the review protocol in Figure 2 is not well-formed either. Specifically, it is not tree-like and it is not well-branched. An equivalent tree-like BP can be obtained by duplicating events as for R_i . However, it is not possible to obtain an equivalent well-branched BP with our current definition.

Finally, we show that well-formedness is retained after a reduction.

Lemma 60. *Let R be a BP and let $R \xrightarrow{\ell} R'$. If R is well-formed then so is R' .*

Proof sketch. We use that the components of R' are subsets of, or derived from (in the case of the branching structure), the components of R . It then follows that a violation of one of the properties in R' would also invariably lead to a violation of one of the properties in R . \square

5.3. Realisability proof

To prove that R 's protocol is realisable, we have to show the existence of a bisimilar composition of local BPs and buffers. To do this, we define a canonical decomposition of R by combining our previously defined projections with a buffer construction, and we prove that this canonical decomposition

is bisimilar to R . The (re)construction of the buffer contents of channel \mathbf{ab} based on R , written $\text{buff}_{\mathbf{ab}}(R)$, and the canonical decomposition of R , written $cd(R)$, are defined below.

The buffer construction $\text{buff}_{\mathbf{ab}}(R)$ gathers the receive events in R that have no preceding matching send event. We infer that, since the send has already been fired and the receive has not, the message must be in transit.

Definition 61 (Buffer construction). Let R be a BP. Let \mathbf{a} and \mathbf{b} be agents in R . Let ε be the empty word.

$$\text{Then } \text{buff}_{\mathbf{ab}}(R) = \begin{cases} \mathbf{x} \cdot \text{buff}_{\mathbf{ab}}(R') & \text{if } R' = R - e \text{ and } \lambda(e) = \mathbf{ab}?\mathbf{x} \\ & \text{and } \forall e': \text{ if } e' \prec e \text{ then } \lambda(e') \neq \mathbf{ab}!\mathbf{x} \\ & \text{and } \forall e', \mathbf{y}: \text{ if } e' \prec e \text{ then } \lambda(e') \neq \mathbf{ab}?\mathbf{y} \\ \varepsilon & \text{otherwise} \end{cases}$$

The corresponding message types are nondeterministically put in some order that respects the order of the gathered receive events — if $e_1 \prec e_2$ then e_1 's message type must precede that of e_2 in the constructed buffer. We note that all unmatched receive events are top-level if R is choreographic, and that the same-channel top-level receive events are totally ordered if R is well-channelled. It follows that, although it may add duplicate messages and is nondeterministic in the general case, $\text{buff}_{\mathbf{ab}}(R)$ does not add duplicate messages and is deterministic if R is well-formed.

Definition 62 (Canonical decomposition). Let R be a BP. Let \vec{R} be such that $R_{\mathbf{a}} = R|_{\mathbf{a}}$ for all \mathbf{a} . Let \vec{b} be such that $b_{\mathbf{ab}} = \text{buff}_{\mathbf{ab}}(R)$ for all \mathbf{ab} . Then $cd(R) = \langle \vec{R}, \vec{b} \rangle$ is the *canonical decomposition* of R .

To prove that a well-formed R is bisimilar to $cd(R)$, we define the relation $\mathcal{R} = \{ \langle R, \langle \vec{R}^\dagger, \vec{b} \rangle \rangle \mid \langle \vec{R}^\dagger, \vec{b} \rangle \sim \langle \vec{R}, \vec{b} \rangle = cd(R) \}$ and we prove that \mathcal{R} is a bisimulation relation (Theorem 70). Note that the vector of buffers \vec{b} is the same across the definition; we only allow leeway in the vector of local BPs. The proof consists of the two parts mentioned at the start of this section. Given that $\langle R, \langle \vec{R}^\dagger, \vec{b} \rangle \rangle \in \mathcal{R}$, if one can make some reduction then the other must be able to make the same reduction such that the resulting configurations are again related by \mathcal{R} (Theorem 66, Theorem 67). Additionally, if one can terminate then so should the other (Theorem 68, Theorem 69).

The reason that \mathcal{R} is not simply the set of all $\langle R, cd(R) \rangle$ is that a reduction from $cd(R)$ may not always result in $cd(R')$ for some R' . This is because

choices are only resolved in the BP of the agent causing the reduction. For example, consider BP R_i in Figure 14. Upon Alice sending **yes** the global BP would resolve the choice for both agents simultaneously. However, upon Alice sending **yes** in the canonical decomposition the projection on Bob remains unchanged and still contains receive events for both **yes** and **no**. Since **yes** has been added to the buffer from Alice to Bob, we know that Bob will eventually have to pick the branch containing **yes** — after all, there is no **no** to receive. In other words: this configuration is bisimilar to the canonical decomposition of the resulting global BP, in which the choice has also been resolved for Bob. If there were also some additional **no** being sent from Alice to Bob, e.g., if we replace the messages **int** in R_i with **no**, then R_i being well-channeled and the buffers being ordered would still ensure that we can safely resolve Bob's choice. This crucial insight is formally proven in Theorem 63.

Lemma 63. *Let R be a well-formed BP. Let $\langle \vec{R}, \vec{b} \rangle = cd(R)$. Let ℓ be some label and let $\mathbf{a} = \text{subj}(\ell)$. If $R \xrightarrow{\ell} R'$ and if $\langle \vec{R}, \vec{b} \rangle \xrightarrow{\ell} \langle \vec{R}[R'_{\mathbf{a}}/R|_{\mathbf{a}}], \vec{b}^\dagger \rangle$ and if $R'_{\mathbf{a}} = R'|_{\mathbf{a}}$, then $\langle \vec{R}[R'_{\mathbf{a}}/R|_{\mathbf{a}}], \vec{b}^\dagger \rangle \sim \langle \vec{R}', \vec{b}' \rangle = cd(R')$.*

Proof sketch. If $\ell = \mathbf{ba}?\mathbf{x}$ for some \mathbf{b}, \mathbf{x} then it follows from the well-formedness of R that $R' = R - e$ and the remainder is straightforward. The same is true if $\ell = \mathbf{ab}!\mathbf{x}$ and e is top-level, i.e., $e \in R.\mathcal{B}$.

Otherwise it follows from the well-formedness of R that e is a minimal send event in one of the options of a top-level choice, i.e., $e \in \mathcal{B} \text{ opt } \mathcal{C} \in R.\mathcal{B}$ for some \mathcal{B}, \mathcal{C} , and $R' = R|_{(R.\mathcal{B} \setminus \mathcal{C}) \cup (\mathcal{B} - e)}$. We proceed to show that the set of unmatched receive events in R' is exactly that of R with the addition of the one corresponding to e , and then $\vec{b}' = \text{add}(\mathbf{ab}!\mathbf{x}, \vec{b}) = \vec{b}^\dagger$. It follows that $\langle \vec{R}[R'_{\mathbf{a}}/R|_{\mathbf{a}}], \vec{b}^\dagger \rangle = \langle \vec{R}[R'_{\mathbf{a}}/R|_{\mathbf{a}}], \vec{b}' \rangle$. For the projections, we proceed in two steps:

- First we observe that, since R is well-branched, $\mathcal{B}'|_{\mathbf{c}} = \mathcal{B}|_{\mathbf{c}}$ for all $\mathcal{B}' \text{ opt } \mathcal{C}$ and for all $\mathbf{c} \neq \mathbf{a}, \mathbf{b}$. It follows that $R|_{\mathbf{c}} \sim R'|_{\mathbf{c}}$, and then $\langle \vec{R}[R'_{\mathbf{a}}/R|_{\mathbf{a}}], \vec{b}' \rangle \sim \langle \vec{R}'[R|_{\mathbf{b}}/R'|_{\mathbf{b}}], \vec{b}' \rangle$. Note that the projection on \mathbf{a} is $R'|_{\mathbf{a}}$ and the projection on \mathbf{b} is $R|_{\mathbf{b}}$ on both sides, and the projection on every other \mathbf{c} is bisimilar.
- Next we show that, with the new message added to the buffer, no event can ever fire in $R|_{\mathbf{b}}$ in any other option of \mathcal{C} than \mathcal{B} . It follows that we can discard all other options, and then $\langle \vec{R}'[R|_{\mathbf{b}}/R'|_{\mathbf{b}}], \vec{b}' \rangle \sim \langle \vec{R}', \vec{b}' \rangle = cd(R')$. \square

To satisfy the preconditions of [Theorem 63](#), we additionally prove that R 's reductions can be mirrored by its projection on the reduction label's subject ([Theorem 64](#)) and dually that the reductions of R 's canonical decomposition can be mirrored by R ([Theorem 65](#)). Their proofs rely on the observations that the corresponding event e must be minimal both in R and in $R|_{\mathbf{a}}$, and that the branching structures of the two are the same (modulo discarded leaves). It then follows that the same refinement enables e in both R and $R|_{\mathbf{a}}$.

Lemma 64. *Let R be a tree-like BP. If $R \xrightarrow{\ell} R'$ and $\mathbf{a} = \text{subj}(\ell)$ then $R|_{\mathbf{a}} \xrightarrow{\ell} R'|_{\mathbf{a}}$.*

Lemma 65. *Let R be a well-channeled, tree-like and choreographic BP. Let $\langle \vec{R}, \vec{b} \rangle = cd(R)$. If $\langle \vec{R}, \vec{b} \rangle \xrightarrow{\ell} \langle \vec{R}[R'_{\mathbf{a}}/R|_{\mathbf{a}}], \vec{b}' \rangle$ then $R \xrightarrow{\ell} R'$ for some R' such that $R'_{\mathbf{a}} = R'|_{\mathbf{a}}$.*

Finally, we bring everything together and prove the four necessary steps for bisimulation in the lemmas below, culminating in [Theorem 70](#). The proof for [Theorem 66](#) uses [Theorem 64](#) to show the preconditions of [Theorem 63](#) and then applies the latter. This gives us $cd(R) \xrightarrow{\ell} cd(R') \sim cd(R')$, and since $\langle \vec{R}^\dagger, \vec{b}^\dagger \rangle \sim cd(R)$ the result is then straightforward. The proof for [Theorem 67](#) is analogous but uses [Theorem 65](#). The proofs for [Theorem 68](#) and [Theorem 69](#) respectively use [Theorem 51](#) and [Theorem 52](#) to show that, if one can terminate by refining to the empty set, then so must the other.

Lemma 66. *Let $\langle R, \langle \vec{R}^\dagger, \vec{b}^\dagger \rangle \rangle \in \mathcal{R}$. If R is well-formed and $R \xrightarrow{\ell} R'$ then there exist \vec{R}^\ddagger and \vec{b}^\ddagger such that $\langle \vec{R}^\dagger, \vec{b}^\dagger \rangle \xrightarrow{\ell} \langle \vec{R}^\ddagger, \vec{b}^\ddagger \rangle$ and $\langle R', \langle \vec{R}^\ddagger, \vec{b}^\ddagger \rangle \rangle \in \mathcal{R}$.*

Lemma 67. *Let $\langle R, \langle \vec{R}^\dagger, \vec{b}^\dagger \rangle \rangle \in \mathcal{R}$. If R is well-formed and $\langle \vec{R}^\dagger, \vec{b}^\dagger \rangle \xrightarrow{\ell} \langle \vec{R}^\ddagger, \vec{b}^\ddagger \rangle$ then there exists R' such that $R \xrightarrow{\ell} R'$ and $\langle R', \langle \vec{R}^\ddagger, \vec{b}^\ddagger \rangle \rangle \in \mathcal{R}$.*

Lemma 68. *Let $\langle R, \langle \vec{R}^\dagger, \vec{b}^\dagger \rangle \rangle \in \mathcal{R}$. If R is well-formed and $R \downarrow$ then $\langle \vec{R}^\dagger, \vec{b}^\dagger \rangle \downarrow$.*

Lemma 69. *Let $\langle R, \langle \vec{R}^\dagger, \vec{b}^\dagger \rangle \rangle \in \mathcal{R}$. If R is well-formed and $\langle \vec{R}^\dagger, \vec{b}^\dagger \rangle \downarrow$ then $R \downarrow$.*

Theorem 70. *Let R be a BP. If R is well-formed and $\text{buff}_{\mathbf{ab}}(R) = \varepsilon$ for all \mathbf{ab} then the protocol represented by R is realisable.*

Proof. It follows from [Theorem 66](#), [Theorem 67](#), [Theorem 68](#) and [Theorem 69](#) that \mathcal{R} is a bisimulation relation [31]. Specifically, it then follows that $R \sim cd(R)$. Then, since $\text{buff}_{\mathbf{ab}}(R) = \varepsilon$ for all \mathbf{ab} , by [Theorem 48](#) the protocol represented by R is realisable. \square

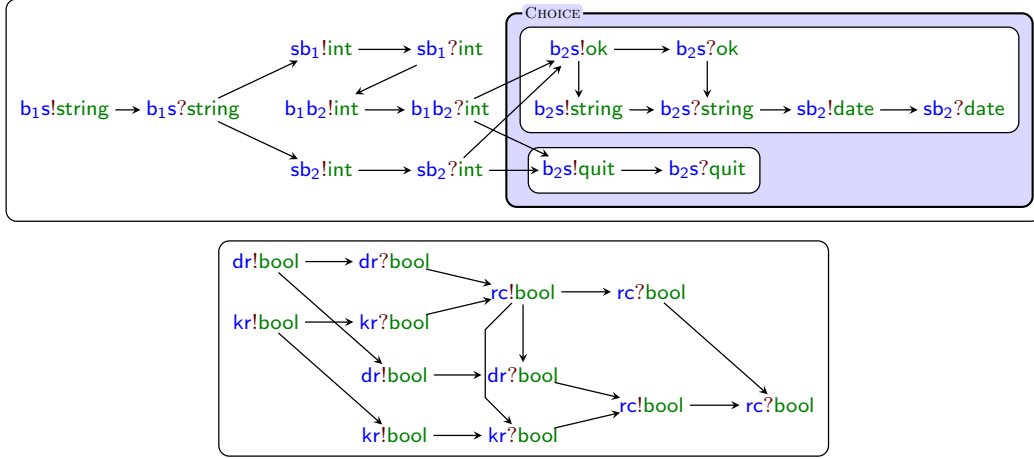


Figure 16: BPs representing the **two-buyers-protocol** (top) and two iterations of the **simple streaming protocol** (bottom) [3].

5.4. Examples

Finally, we briefly look at two example protocols used by Honda et al. [3]. Both are depicted as BPs in Figure 16.

The **two-buyers-protocol** (top) features Buyer 1 and Buyer 2 (b_1, b_2) who wish to jointly buy a book from Seller (s). Buyer 1 first sends the title of the book (**string**) to Seller, Seller sends its quote (**int**) to both Buyer 1 and Buyer 2, and Buyer 1 sends Buyer 2 the amount they can contribute (**int**). Buyer 2 then notifies Seller whether they accept (**ok**) or reject (**quit**) the quote. If they accept, they also send their address (**string**), and Seller returns a delivery date (**date**).

The **simple streaming protocol** (bottom) features Data Producer (d) and Key Producer (k), who continuously respectively send data and keys (both **bool**) to Kernel (r). Kernel performs some computation and sends the result (**bool**) to Consumer (c). The protocol in Figure 16 shows two iterations of this process.

Both BPs are well-formed, and hence the protocols are realisable. We note that, as in the paper by Honda et al., further communication between Buyers 1 and 2 has been omitted in the two-buyers-protocol. Since this is bound to be different in the case of acceptance and rejection, the resulting BP would not be well-branched and thus not well-formed — though still realisable. We discuss relaxed well-branchedness conditions in Section 7. Also note that the **ok** and address (**string**) messages are sent sequentially; **sending these in par-**

`allel` would violate well-channelledness and make the protocol unrealisable with ordered buffers. The same is true for the streaming protocol: the two iterations are composed sequentially and `doing so concurrently` would violate well-channelledness and result in unrealisability. The size of the BP for the streaming protocol scales linearly with the number of depicted iterations; showing all (infinitely many) iterations would require an infinitely large BP. We briefly touch upon infinity in Section 8.

6. Tool: B-Pomset Encoder

We developed a companion tool to simulate BPs and analyse their applications to choreographies. The tool is open-source (<https://github.com/arcalab/choreo/tree/b-pomset>), developed in Scala and compiled into JavaScript (JS). A snapshot of a compiled JS can be executed from an internet browser at <http://lmf.di.uminho.pt/b-pomset/>.

The tool fulfils 3 main objectives: (1) *internal validation*, providing the authors early insights over alternative notions of well-formedness, (2) *dissemination*, to better explain the propose analysis using an interactive environment, and (3) *algorithmic insights* of well-formedness (cf. Section 5.2), by transforming the declarative definitions into concrete algorithms and identifying possible bottlenecks. This section focuses on the last two objectives, i.e., on *how to use* the online tool, and on *how to calculate* well-formedness.

6.1. Using the B-Pomset Encoder

A screenshot of our analyser of BPs can be found in Figure 17. The tool displays a collection of widgets, including the “Choreography” widget with an editor where the user describes the BP to be analysed. Each widget can be expanded or collapsed by clicking on the widget title; e.g., in the screenshot the widget “Choreography” is expanded and the widget “Global LTS” is collapsed. Clicking the refresh button at the top-right of the “Choreography” widget (or pressing shift-enter) updates all expanded widgets.

BPs are described using choreographies (Section 4), and are visualised in widget “Global B-Pomset” after being encoded. Since some BPs cannot be described by choreographies, one can also write actions explicitly (e.g., “`a!b:x`” represents the action `ab!x`) and can extend the causality relation (e.g., “[1->3]” denotes that the 1st event must precede the 3rd event), where 1 and 3 are the event identifiers depicted in the BP diagram. The BP R_i in Figure 14 can be written both as “`(a->b:yes + a->b:no) ; a->b:int`” and as

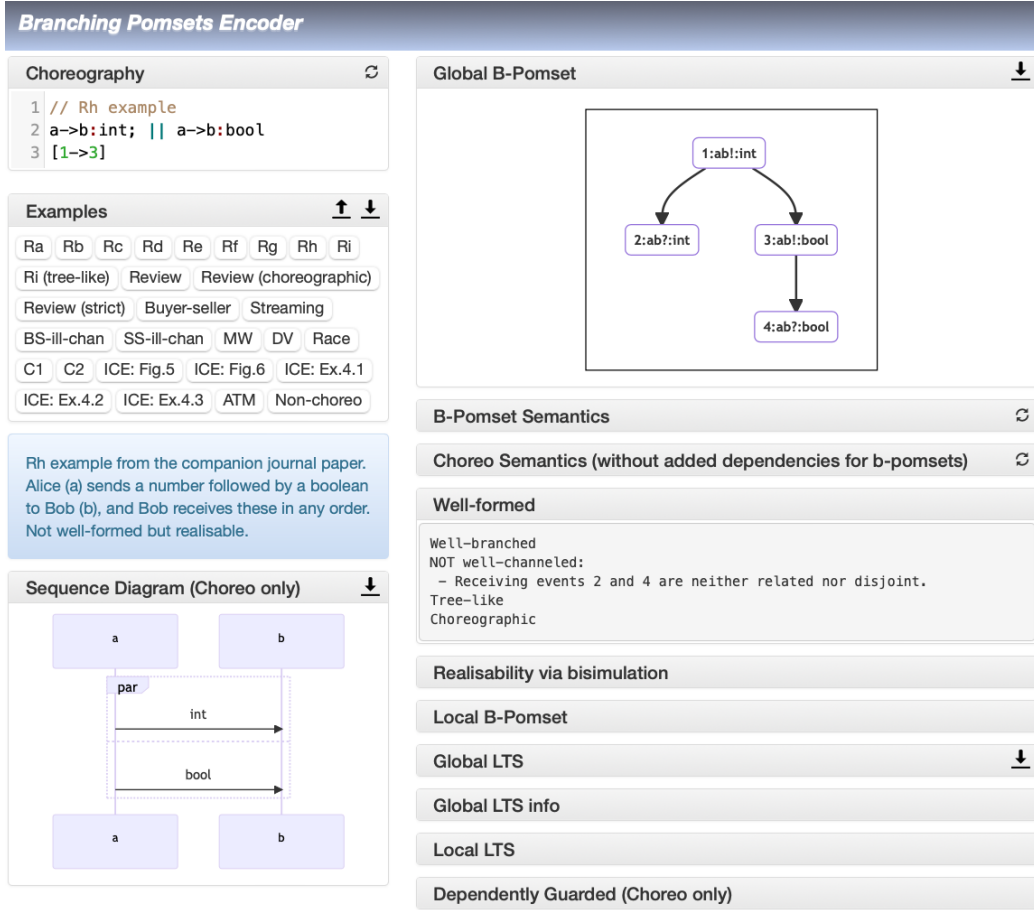


Figure 17: Screenshot of the B-Pomset Encoder tool.

“(a!b:yes|a?b:yes)+(a!b:no|a?b:no))||a!b:int||a?b:int[1->2,1->5, 2->6,3->4,3->5,4->6,5->6]” A list of different examples can be found in the “Examples” widget; clicking any of them will load this example to the “Choreography” widget.

The remaining widgets provide different analyses, including the following:

- “Global B-Pomset” draws the encoded BP from “Choreography”;
- “B-Pomset Semantics” allows a step-by-step exploration of the reduction semantics of an encoded BP (Figure 3), drawing the intermediate BPs;
- “Choreo Semantics (...)” allows a step-by-step exploration of the reduction semantics of the provided choreography (Figure 10);

- “Well-formed” checks if the BP is well-formed; it states for each well-formedness property whether it holds and gives a counterexample when it does not;
- “Realisability via bisimulation” searches for a bisimulation between the global LTS and the composition of all local LTSs (Theorem 47); this acts as our ground truth for realisability (Theorem 48), but it is often infeasible due to state explosion;
- “Local B-Pomset” draws the projections of the BP to each agent;
- “Global LTS” and “Local LTS” draw state machines using the semantics of BPs (Figure 3) applied to the global and to the projected BPs, respectively (bounding the number of states to a maximum value);
- “Sequence Diagram (...)” draws a sequence diagram representing the choreography (ignoring the extended causality arrows used to produce non-choreographic BPs).

Remark (loops). Our implementation targets only *finite* BPs. Hence choreographies with loops are not covered by our analysis. However, the current version includes experiments with loops. I.e., some analyses will produce some results for BPs with special branches marked as loops. The theory for this extension is still under investigation and not documented in this paper.

6.2. Realising well-formedness

The (open) source code of our implementation of well-formedness can be found on GitHub.¹¹ This implementation realises the declarative definitions of each of the four sub-properties of well-formedness (Theorem 58).

Our implementation provides some insights regarding the complexity of verifying these four sub-properties. Recall that analysing realisability, as defined in Theorem 48, requires traversing the full state-space of our semantics, which explodes exponentially in the presence of concurrent events. Our implementation avoids expanding the full state-space by combining multiple traversals over the BP graph structure. Below we sketch the implementation of each of these four sub-properties, providing evidences that it is less complex than traversing all states.

¹¹<https://github.com/arcalab/choreo/blob/b-pomset/src/main/scala/choreo/realisability/WellFormedness.scala>

- **Well-branched.** (Theorem 54) Our tool finds every choice with branches \mathcal{B}_1 and \mathcal{B}_2 . It proceeds by: (1) recursively checking if both branches are well-branched, (2) finding all *leaders* in each branch, i.e., events with no predecessor in the same branch (3) collecting sending and receiving agents involved, and (4) verifying that only 1 leading sender and receiver is found (i.e., agents involved in events with no predecessors in the branch), that they are the same in both branches, and that they have different messages. Furthermore, (5) each of these branches is projected to all remaining agents, and (6) corresponding projections from both branches are compared using graph isomorphism.¹²

The causality relation is modelled as hash-map from events to their predecessors (not being minimal nor transitive closed). Most operations are linear on the number of events, whereas the most complex operations are projecting both branches to all agents that are not in the leading events (5), and check if each projection is isomorphic to its neighbour branch (6).

- **Well-channelled** (Theorem 55) Our tool starts by collecting, for each pair of agents a and b , the sets of channels $EM_{ab}^!$ and $EM_{ab}^?$, each consisting of events and messages (e, m) sent by a and received by b , respectively. It then proceeds in two parts.

Firstly, for every distinct pair $(e_1, m_1), (e_2, m_2) \in EM_{ab}^!$, it checks if e_1 and e_2 are either related ($e_1 \preceq e_2$ or $e_2 \preceq e_1$) or are exclusive (in opposing branches of a choice). It also performs the same check for $EM_{ab}^?$.

Secondly, for each set $EM_{ab}^?$, it collects every event e paired with a direct predecessor e' , and for every distinct pair (e'_1, e_1) and (e'_2, e_2) from this set it checks if $e_1 \preceq e_2 \Rightarrow e'_1 \preceq e'_2$.

In these calculations, checking $e \preceq e'$ requires traversing the predecessor graph from e' until it finds e , and the *direct* predecessor relation is computed once by discarding redundant arcs in the predecessor relation. A worst case scenario is when the same channel appears in many places, leading to large $EM_{ab}^!$ and $EM_{ab}^?$ sets. In turn, this would be a

¹²We used an existing algorithm for finding isomorphisms of acyclic graphs [35] applied to the predecessor relation.

best case scenario to check well-branchedness, where fewer projections and graph isomorphisms checks would be needed.

- **Tree-like** (Theorem 56) Our tool starts by computing the successor relation by inverting the predecessor relation. It then checks, for every branch \mathcal{B} of every choice if the set of all successors of its events includes elements outside that branch.
- **Choreographic** (Theorem 57) Our tool pre-computes the successor relation (inverting the predecessor one), and for every event e proceeds in three parts.

















Firstly, it traverses the predecessor until it finds, in every branch of this traversal, either (1) an event labelled by a sending action that matches the action of e , or (2) an event with the same subject.

Secondly, when e is labelled by some $ab?m$ and is inside a branch of a choice, it checks if any of the neighbour events in the same branch (not further nested) is also a predecessor and labelled by a matching $ab!m$.

Thirdly, when e is labelled by some $ab!m$, it uses the successor relation to traverse all its successors. When finding an event with a matching $ab?m$ our tool collects it paired with the set of events $ab!m$ passed by during the traversal. It then checks if exactly one of these pairs has exactly e as its only predecessor.

This informal explanation provides an overview of the set of traversals over the BP graph required to verify the four conditions that guarantee well-formedness. It also highlights the compactness of the formal definitions with respect to their realisation. Intuitively, we expect our implementation to have polynomial (time) complexity, corresponding to situations where, for every event, we need to reason over its neighbours in a branch or its predecessors (as in the choreographic check). Furthermore, checking if a BP is well-channelled, when all its N events are labelled by sending actions $ab!m_i$ over the same agents and different messages m_i , would require comparing all pairs of messages (complexity $O(N^2)$). Each of these comparisons could further require traversing the predecessor relation, which in the worst case could mean traversing the N events. In any of these scenarios, the complexity is well below the exponential complexity required to analyse realisability via bisimilarity, and can be validated by experimenting with the online tool.

Table 2: Time (μs) to infer well-formedness and realisability via bisimulation of the examples in this paper; values with **filled backgrounds** represent tests that reject the BP.

pg.	BP	Well-formed				Total	Realisable	Ratio
		WC	WB	TL	CH			
3	c_{fst} 	364	277	53	413	1109 \pm 222	61013 \pm 73838	1.81%
4	c_{snd}^{strict} 	554	353	154	614	1677 \pm 936	105189 \pm 34176	1.59%
10	$c_{snd}^{lenient}$ 	536	417	73	572	1602 \pm 370	114348 \pm 9500	1.4%
35	c_1 	47	77	30	115	272 \pm 237	6012 \pm 2814	4.52%
37	R_c 	39	81	39	116	277 \pm 113	2308 \pm 865	12.0%
37	R_d 	50	161	57	260	530 \pm 174	7000 \pm 565	7.57%
40	R_e 	1	52	1	46	102 \pm 128	1262 \pm 325	8.08%
46	R_f 	41	111	32	89	276 \pm 277	2360 \pm 193	11.69%
46	R_g 	69	112	23	114	321 \pm 165	6132 \pm 765	5.23%
46	R_h 	1	30	1	38	73 \pm 142	743 \pm 542	9.82%
46	R_i 	54	96	43	70	266 \pm 165	2359 \pm 425	11.27%
50	R_i tree-like 	82	165	42	121	413 \pm 344	2508 \pm 487	16.46%
54	2-buyers 	183	294	55	606	1141 \pm 364	26746 \pm 1963	4.26%
54	streaming 	2	147	1	202	356 \pm 577	54058 \pm 6714	0.65%
54	2-buyers ill 	187	46	46	450	732 \pm 114	25099 \pm 973	2.91%
55	streaming ill 	1	20	1	45	70 \pm 81	8638681 \pm 605334	0.0%

6.3. Performance evaluation

We executed most examples mentioned in this paper with our tool and measured the time to compute well-formedness and realisability (via bisimulations). We used the same source code, but compiled to and executed from a Java Virtual Machine, instead of using JavaScript. The results are presented in Table 2, using the abbreviations WC (well-channeled), WB (well-branched), TL (tree-like), and CH (choreographic). Each measurement was repeated ten times — we write $a \pm \delta$ to capture both the average a and the greatest deviation δ . The ratio in the last column divides the time to check well-formedness by the time to check realisability.

The experiments presented here do not aim at exploring corner cases regarding best- and worst-scenarios for well-formedness. Instead, the goal is to illustrate that checking well-formedness does not suffer the same state explosion problem as a search for a bisimulation over the state-space.

Based on the results in Table 2, we draw three main conclusions.

1. The ratios in the rightmost column indicate that, in all examples, check-

ing well-formedness is one order of magnitude faster than checking bisimulation-based realisability (ratio between 10%-100%), two orders (1%-10%), or even three orders (0.1%-1%). Furthermore, we note that the speedup of checking well-formedness tends to be larger in cases where checking realisability is relatively expensive (i.e., relatively high values in the “Realisable” column). This suggests that checking well-formedness scales better than checking realisability.

2. In three examples, the check for well-formedness was negative, while the check for realisability was positive (c_{snd}^{strict} , $c_{snd}^{lenient}$, and R_i). This demonstrates that well-formedness is a sound, but incomplete, condition that conservatively approximates realisability. An important piece of future work is to make well-formedness more liberal.
3. Regarding the sub-conditions of well-formedness, checking tree-likeness tends to be the cheapest among the four. Neither one of the three other sub-conditions clearly stands out as being the most expensive one; it tends to depend on the structure of the branching pomset under consideration.

We note that we consider these experimental results preliminary; a more extensive practical evaluation is needed to better understand, e.g., the performance of our approach in corner cases. However, we do believe that [Table 2](#) already gives an encouraging general impression.

7. Related work

Pomsets. Pomsets were initially introduced by Pratt [18] for concurrent models and have been widely used, e.g., in the context of message sequence charts by Katoen and Lambert [29]. Recently Guanciale and Tuosto proposed two semantic frameworks for choreographies, one of which uses sets of pomsets [36]. They also note that the pomset framework exhibits exponential growth in the number of choices in a choreography, and they propose an alternative semantic framework using hypergraphs, which can compactly represent choices. While the hypergraph framework is more compact, their pomset framework is simpler and, they believe, more elegant. We agree with this analysis, and we aim to preserve the simplicity and elegance of the pomset framework by proposing a framework that avoids exponential growth in the number of choices while still being based on pomsets.

Event structures. Section 3 gives a detailed formal comparison of the expressive power of BPs and several classes of ESs from the literature. Event structures (ESs) (resp. labelled ESs) were introduced as a generalisation of posets (resp. pomsets). The main difference is in the added choice mechanism: ESs use a conflict relation to forbid certain pairs of actions of occurring in the same execution, while BPs use an explicit branching structure. Generally speaking, the choice mechanisms of ESs are more expressive than BPs’ branching structure, but the latter’s refinement semantics raise its expressiveness in orthogonal ways. As a result, BPs are incomparable with the most common classes of ESs and are only properly contained in some of the more expressive classes of dynamic event structures.

Choreographies. Choreographies are typically used in a top-down workflow: the developer writes a global view C and decomposes it into its projections, such that the behaviour of C is *behaviourally equivalent* to the parallel composition of its projections. Examples of this approach include workflows based on message sequence charts [1, 2], multiparty session types [3, 4], and choreographic programs [5, 6]. The choreographic language used in this paper assumes asynchronous communication between agents and includes a finite loop operator, borrowing from this literature the same notion of interactions as syntactic atoms and their (parallel, sequential, and choice) composition.

Realisability. Realisability has been well-studied in the last two decades in a variety of settings. For example, Alur et al. study the realisability of message sequence charts [37]. They define the notions of weak and safe realisability of languages, the latter also ensuring deadlock-freedom, and they define closure conditions on languages which they show to precisely capture weakly and safely realisable languages. Lohmann and Wolf define the notion of distributed realisability, where a protocol is distributedly realisable if there exists a set of compositions such that every composition covers a subset of the protocol and the entire protocol is covered by their union [8]. Fu et al. [7], Basu et al. [9], Finkel and Lozes [38] and Schewe et al. [10] all study the realisability of protocols on different network configurations when considering only the sending behaviour — receive events are omitted — showing necessary and/or sufficient conditions and decidability results.

Realisability of pomsets. One major source of inspiration for our work has been the recent work by Guanciale and Tuosto, in which they presented a

realisability analysis based on sets of pomsets [13]. They show how to capture the language closure conditions of Alur et al. [37] directly on pomsets, without having to explicitly compute their language. Typically choreography languages are limited in their expressiveness and any analysis on their realisability is then language-dependent. Both Alur et al. and Guanciale and Tuosto perform a syntax-oblivious analysis, which has the benefit of being applicable to any specification which can be encoded as a set of pomsets, regardless of the specification language. The analysis by Guanciale and Tuosto is at a higher level of abstraction than sets of traces. This allows both for a more efficient analysis and for easier identification of design errors, as these can be identified in a more abstract model.

Our approach is similarly syntax-oblivious, though the analysis itself is based on multiparty session types. A major technical difference is that Guanciale and Tuosto use unordered buffers (e.g., non-FIFO queues) while ours are ordered. For example, the parallel composition of $a \rightarrow b:x$ and $a \rightarrow b:y$ is realisable in the unordered setting and not in the ordered one. The two settings agree on realisability when the two message types are the same (e.g., two concurrent copies of $a \rightarrow b:x$); while Guanciale and Tuosto explicitly note that they support concurrently repeated actions, however, our current well-channelledness condition does not make an exception for these. This is an obvious target for relaxation of our conditions.

Further inspiration for relaxation of well-formedness conditions can be found in an earlier paper by the same authors [36]. In particular their definition of well-branchedness, using ‘active’ and ‘passive’ agents, could serve as a basis for a relaxed version of our own.

A meaningful in-depth comparison between the pomset-based approach by Guanciale and Tuosto [13] and ours, both in terms of expressiveness and efficiency, would first require further development (relaxation) of our conditions and is therefore left for future work. In the meantime, one takeaway from their paper is the performance gain they obtain by lifting the analysis from languages (sets of traces) to a higher level of abstraction, i.e., sets of pomsets. Our hope is that a further performance gain can be obtained by lifting the analysis from sets of pomsets to an even higher level of abstraction (e.g., branching pomsets).

Multiparty session types. The other major source of inspiration for our work is multiparty session types (MPST), introduced by Honda et al. [3]. Specifically:

- Our well-branchedness condition corresponds to the branching syntax of global types in MPST and its definition of projection. Branching in MPST is done on the label of a single message, and the projection on agents uninvolved in the choice is only defined if it is the same in every branch.
- Our well-channeledness condition corresponds to the principle that same-channeled actions should be ordered. We note that our condition is more liberal: we prohibit concurrent sends or receives on the same channel, while in MPST the projection of a parallel composition on an agent is undefined if the agent occurs in both parallel branches (even if those branches' channels are disjoint).
- Our tree-likeness condition follows from the syntax of global types in MPST, which uses a prefix operator rather than a more general sequential composition. As a consequence all global types are tree-like. The same is true for other languages that use a prefix operator and not sequential composition, such as CCS [39] and π -calculus [40].

Since its introduction, various papers have addressed the conservativeness of the well-branchedness condition in MPST. One line of research relaxes the condition by using a merge operation to allow all agents to have different behaviour in different branches, as long as they are timely informed of the choice [41, 42, 43]. Another line relaxes the condition by allowing different branches to start with different receivers, rather than enforcing the same receiver in every branch [12, 44, 45, 46, 47]. These results may also serve as inspiration for relaxations of the well-branchedness condition on branching pomsets.

While our current conditions broadly correspond with well-formedness in MPST, we believe that our approach offers three advantages. First, as discussed before, it is syntax-oblivious, meaning it is not only applicable to MPST but to any specification which can be encoded as a branching pomset. Second, we believe that branching pomsets have the potential to be more expressive than global types in MPST. As mentioned above, our well-channeledness condition is already more liberal than the one in MPST. We have described various sources of possible relaxations for our well-branchedness condition, both in the MPST and in the pomset literature. Lastly, we conjecture that our tree-likeness condition is needed to

simplify our proofs, and that it is possible — though more complex — to prove realisability without it (or at least with a relaxed version of it).

8. Conclusion

In this section we briefly summarise the paper and discuss (1) possible improvements on BPs in general, (2) ideas for future work on event structures, (3) relaxations of our well-formedness conditions on BPs for choreographies, (4) and ideas for future work on the tool.

8.1. Summary

- In [Section 2](#) we have defined a new model for communication protocols, branching pomsets ([Theorem 3](#)), which can compactly represent both concurrency and choices, and have defined its semantics ([Figure 3](#)).
- In [Section 3](#) we have compared the expressive power of BPs with several classes of event structures from the literature. [Figure 7](#) summarises our findings: we have shown that BPs are incomparable with prime ESs, with extended bundle ESs and with growing and shrinking causality ESs; we conjecture that BPs describe proper subsets of a variant of dynamic causality ESs; we prove their inclusion in ESs for resolvable conflict.
- In [Section 4](#) we have defined a choreographic language and its operational semantics ([Figures 9 and 10](#)) using the weak sequential composition and partial termination of Rensink and Wehrheim [\[30\]](#), which is novel in the context of choreographies. We have shown that we can use BPs to model choreographies ([Figure 12](#)) and that this model is behaviourally equivalent (bisimilar) to the operational semantics ([Theorem 46](#)).
- In [Section 5](#) we have defined well-formedness conditions on BPs for choreographies ([Theorem 58](#)) and have proven that a well-formed BP represents a realisable protocol ([Theorem 70](#)). Since these conditions are defined on BPs, they are generic: we can use them to reason about choreographies in the language defined in [Section 4](#) and any other language that can be encoded in BPs, without having to develop a specific analysis for the language in question. Our conditions are sufficient but not necessary, we discuss possible relaxations later in this section.

- In [Section 6](#) we have presented our prototype tool and described the algorithms used to implement well-formedness. Our tool is compiled to JavaScript that can be executed by any recent internet browser; a screenshot can be found in [Figure 17](#). For any given BP, our tool can verify well-formedness, apply the reduction rules, and verify realisability using bisimulation, among other analysis.

8.2. Discussion regarding branching pomsets

n-ary choices. Our branching structures only supports *n*-ary choices as a derived concept, through nested binary choices. This matches the structure of choreographies, but it would be more natural to represent $c_1 + (c_2 + c_3)$ as a single choice between the pomsets $\llbracket c_1 \rrbracket$, $\llbracket c_2 \rrbracket$ and $\llbracket c_3 \rrbracket$ instead of as two nested binary choices. However, supporting arbitrary *n*-ary choices also requires some thought about how to change the rules for refinement ([Figure 3a](#)), in particular CHOICE. A naive change would be to simply have this rule use $i \in \{1, \dots, n\}$ and $\{\{\mathcal{B}_1, \dots, \mathcal{B}_n\}\}$ instead of its current binary rules, but this is not sufficient as this naive *n*-ary choice would not be equivalent to the same branches composed as nested binary choices. For example, $c_1 + (c_2 + c_3)$ can partially terminate to $c_1 + c_2$ and its interpretation as a BP can refine to $\llbracket c_1 + c_2 \rrbracket$, but a BP whose branching structure consists of a single ternary choice $\{\{\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3\}\}$ would not be able to refine to $\{\{\mathcal{B}_1, \mathcal{B}_2\}\}$ as the rules would only allow it to either refine all of its branches or discard all but one of them. Properly supporting *n*-ary choices would thus also require a new rule that allows $\{\{\mathcal{B}_1, \dots, \mathcal{B}_n\}\}$ to refine a choice between an arbitrary (non-empty) subset of its branches.

Partial order. In [Theorem 3](#), \preceq is defined as a relation on events such that its transitive closure is a partial order, rather than \preceq being a partial order itself as \leq is in traditional pomsets. This is necessary because two events may be either related or unrelated depending on the resolution of certain choices. For example, consider the BP R_c in [Figure 11](#). The events labelled $ab!x$ and $cd!x$ are related if the choice's upper branch is taken and unrelated otherwise. It should thus be possible to enable $cd!x$ by discarding the choice's upper branch. In our current rules $cd!x$ only has a direct dependency on $bc?x$ and therefore discarding $bc?x$ will make $cd!x$ minimal. However, if \preceq were a partial order, then $cd!x$ would also have direct dependencies on $ab!x$, $ab?x$ and $bc!x$ and, since the first two events are not part of the choice, there would be no refinement that enables $cd!x$.

In general, if $R_1 \sqsupseteq R'_1$ and $R_2 \sqsupseteq R'_2$ then if \preceq would be a partial order it would not necessarily be true that $R_1 ; R_2 \sqsupseteq R'_1 ; R'_2$: $R_1 ; R_2$ may contain dependencies obtained by transitivity which would still be present in the refinement but which cannot be derived in $R'_1 ; R'_2$. Castellani and Zhang note the same property in the context of flow event structures [48].

Loops. In Figure 12 a loop c^* is encoded by infinitely unfolding it. As such, BPs do not currently provide a finite representation of infinite choreographies. We envision three possible directions. One possibility would be to add an explicit repetition construct to the branching structure (e.g., change the second grammatical rule to $\mathcal{C} = e \mid \{\mathcal{B}_1, \mathcal{B}_2\} \mid \mathcal{B}^*$) and expand the semantics and proofs accordingly. Alternatively, a solution might be found in the extension from message sequence charts (MSCs) to MSC-graphs [1]. A similar extension could be developed for branching pomsets, where they are sequentially composed in a (possibly cyclic) graph. Finally, it may be possible to leverage the recently introduced pomset automata [49].

8.3. Discussion regarding event structures

Relation between BPs and ESs. Figure 7 shows the established inclusions and non-inclusions between BPs and ESs. Naturally, formally proving or disproving the inclusion of BPs in our variant of dynamic causality ESs (Theorem 35) would complete the picture. Another logical comparison to make would be with the original semantics of dynamic causality ESs, which are not covered in this paper. Additionally, it may be interesting to further study the difference in expressiveness between BPs and the classes of ESs. As demonstrated by Theorem 28, (prime) ESs can describe more complex choices than BPs. A way to close or overcome the gap would be to lift the requirement on a BP's branching structure that all events must occur in it exactly once, thus allowing overlapping boxes in the graphical representation. This would also require changes to the refinement rules. As the branching structure could then encode a (symmetric) conflict relation, this would invalidate Theorem 28. It is unclear precisely how it would affect the other relations.

Well-formedness for event structures. Because of the close relation between BPs and ESs, it is tempting to try to adapt our well-formedness conditions to the latter. We believe, however, that this is not straightforward. Our well-branchedness condition in particular reasons over choices and their

branches, which poses a problem in two ways: (1) ‘choices’ in the sense of well-branchedness are only represented implicitly in ESs, which would require extracting them first, and (2) as stated before, ESs can describe more complex choices than BPs, which would require a different way of reasoning about them. A more promising approach might be to in some way apply the notion of ‘active’ and ‘passive’ agents in the well-formedness conditions for pomsets by Guanciale and Tuosto [36] to ESs. Well-formedness of ESs is also discussed by Castellani et al. in a recent paper on ES semantics for multiparty sessions [15], where they also reference the conditions by Guanciale and Tuosto.

8.4. Discussion regarding well-formedness conditions

We have discussed several possible relaxations of well-branchedness in Section 7. We now discuss the other three conditions.

Well-channeledness. The BP R_h in Figure 14 is not well-channeled since the events labelled with $ab?int$ and $ab?bool$ are unordered. It is unrealisable because a local system will force the int to be received before the $bool$ while the global BP allows a different order. However, in this case one may take the view that the problem is not the local system being too strict, but rather the global BP being too liberal in an environment with ordered buffers: it should then in some way allow a user to specify just the two acceptable orderings without having to resort to adding a choice between the two and duplicating all events in the BP. Therefore, instead of changing the well-channeledness condition, another avenue would be to change the reduction rules in for branching pomsets themselves (Figure 3) and specifically adapt them to ordered buffers. This could be done in such a way that reducing R_h by firing $ab!int$ then automatically adds a dependency from $ab?int$ to $ab?bool$. This might allow the well-channeledness condition to be significantly relaxed or to possibly be removed altogether.

Tree-likeness. Having the assumption of tree-likeness simplifies our proofs. It is our aim to eventually relax or even remove it and still prove realisability, but this will require a significantly more complex proof. We have noted in Section 7 that global types in MPST and expressions in, e.g., CCS and the π -calculus, are tree-like by default. Conceptually a non-tree-like BP could potentially be turned into an equivalent (i.e., bisimilar) tree-like one by distributing the offending events over the branches of the involved choice.

For example, consider the BP R_i in Figure 14. By duplicating ab!int and ab?int and adding a copy of each with the relevant dependencies to each of the two branches of the choice, we obtain a bisimilar but now tree-like (and well-formed) BP. A more general scheme may be developed based on versions of the CCS expansion theorem [50, 51]. However, regaining expressiveness at the cost of duplicating events effectively negates the benefits of using BPs in the first place.

Choreographicness. The choreographicness condition works well for BPs derived from choreographies, but could eventually be relaxed to also allow more general BPs. Choreographicness forces the two events in a send-receive pair to be siblings in the branching structure. For example, in BP R_i in Figure 14 it would be disallowed to distribute ab!int over the choice while leaving ab?int outside, even though the resulting BP would be bisimilar to R_i .

8.5. Future work regarding the tool

BP variations. Choreographies with loops generate BPs with infinite events. Our current implementation includes some experimental support to represent loops symbolically, which we plan to further investigate. Other variations of BPs, e.g., with n -ary choices mentioned above, could also be added to the implementation.

Event Structures. Our future work includes possible encoding of BPs as dynamic causality ESs (Theorem 35) and as ESs for resolvable conflict (Theorem 36). These encodings can be also implemented and included in our prototype tool.

API generation. We have investigated how to generate APIs in Scala 3, which can verify compliance at compile time of an implementation with respect to a projected pomset [11]. We believe that this can be further explored in the context of BPs.

Acknowledgments

We thank the anonymous reviewers for their many insightful remarks, as well as for their advices on structuring the presentation of our contributions.

Funding. *Sung-Shik Jongmans*: Netherlands Organisation of Scientific Research: 016.Veni.192.103. *José Proença*: This work was partially supported by National Funds through FCT/MCTES (Portuguese Foundation for Science and Technology), within the CISTER Unit (UIDP/UIDB/04234/2020) and the IBEX project (PTDC/CCI-COM/4280/2021); by the EU/Next Generation, within the Recovery and Resilience Plan, within project Route 25 (TRB/2022/00061 – C645463824-000000063); and by national funds through FCT and European funds through EU ECSEL JU, within project VALU3S (ECSEL/0016/2019 - JU grant nr. 876852). *Ilaria Castellani*: This work was partially funded by the ANR17-CE25-0014-01 CISC project.

References

- [1] ITU, Itu-t recommendation z.120. message sequence chart (MSC) (2011).
- [2] R. Alur, K. Etessami, M. Yannakakis, Realizability and verification of MSC graphs, *Theor. Comput. Sci.* 331 (1) (2005) 97–114. doi:[10.1016/j.tcs.2004.09.034](https://doi.org/10.1016/j.tcs.2004.09.034).
- [3] K. Honda, N. Yoshida, M. Carbone, [Multiparty asynchronous session types](#), in: G. C. Necula, P. Wadler (Eds.), *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008, San Francisco, California, USA, January 7-12, 2008*, ACM, 2008, pp. 273–284. doi:[10.1145/1328438.1328472](https://doi.org/10.1145/1328438.1328472). URL <https://doi.org/10.1145/1328438.1328472>
- [4] K. Honda, N. Yoshida, M. Carbone, Multiparty asynchronous session types, *J. ACM* 63 (1) (2016) 9:1–9:67. doi:[10.1145/2827695](https://doi.org/10.1145/2827695).
- [5] M. Carbone, F. Montesi, Deadlock-freedom-by-design: multiparty asynchronous global programming, in: R. Giacobazzi, R. Cousot (Eds.), *The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '13, Rome, Italy - January 23 - 25, 2013*, ACM, 2013, pp. 263–274. doi:[10.1145/2429069.2429101](https://doi.org/10.1145/2429069.2429101).
- [6] L. Cruz-Filipe, F. Montesi, A core model for choreographic programming, *Theor. Comput. Sci.* 802 (2020) 38–66. doi:[10.1016/j.tcs.2019.07.005](https://doi.org/10.1016/j.tcs.2019.07.005).

- [7] X. Fu, T. Bultan, J. Su, [Conversation protocols: a formalism for specification and verification of reactive electronic services](#), Theor. Comput. Sci. 328 (1-2) (2004) 19–37. doi:[10.1016/j.tcs.2004.07.004](#). URL [https://doi.org/10.1016/j.tcs.2004.07.004](#)
- [8] N. Lohmann, K. Wolf, [Realizability is controllability](#), in: C. Laneve, J. Su (Eds.), Web Services and Formal Methods, 6th International Workshop, WS-FM 2009, Bologna, Italy, September 4-5, 2009, Revised Selected Papers, Vol. 6194 of Lecture Notes in Computer Science, Springer, 2009, pp. 110–127. doi:[10.1007/978-3-642-14458-5_7](#). URL [https://doi.org/10.1007/978-3-642-14458-5_7](#)
- [9] S. Basu, T. Bultan, M. Ouederni, [Deciding choreography realizability](#), in: J. Field, M. Hicks (Eds.), Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2012, Philadelphia, Pennsylvania, USA, January 22-28, 2012, ACM, 2012, pp. 191–202. doi:[10.1145/2103656.2103680](#). URL [https://doi.org/10.1145/2103656.2103680](#)
- [10] K. Schewe, Y. A. Ameur, S. Benyagoub, [Realisability of choreographies](#), in: A. Herzig, J. Kontinen (Eds.), Foundations of Information and Knowledge Systems - 11th International Symposium, FoIKS 2020, Dortmund, Germany, February 17-21, 2020, Proceedings, Vol. 12012 of Lecture Notes in Computer Science, Springer, 2020, pp. 263–280. doi:[10.1007/978-3-030-39951-1_16](#). URL [https://doi.org/10.1007/978-3-030-39951-1_16](#)
- [11] G. Cledou, L. Edixhoven, S. Jongmans, J. Proença, API generation for multiparty session types, revisited and revised using scala 3, in: K. Ali, J. Vitek (Eds.), 36th European Conference on Object-Oriented Programming, ECOOP 2022, June 6-10, 2022, Berlin, Germany, Vol. 222 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, pp. 27:1–27:28. doi:[10.4230/LIPIcs.ECOOP.2022.27](#).
- [12] P. Deniélou, N. Yoshida, [Multiparty session types meet communicating automata](#), in: H. Seidl (Ed.), Programming Languages and Systems - 21st European Symposium on Programming, ESOP 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings,

- Vol. 7211 of Lecture Notes in Computer Science, Springer, 2012, pp. 194–213. doi:[10.1007/978-3-642-28869-2_10](https://doi.org/10.1007/978-3-642-28869-2_10).
URL https://doi.org/10.1007/978-3-642-28869-2_10
- [13] R. Guanciale, E. Tuosto, [Realisability of pomsets](#), J. Log. Algebraic Methods Program. 108 (2019) 69–89. doi:[10.1016/j.jlamp.2019.06.003](https://doi.org/10.1016/j.jlamp.2019.06.003).
URL <https://doi.org/10.1016/j.jlamp.2019.06.003>
 - [14] M. Nielsen, G. Plotkin, G. Winskel, Petri nets, event structures and domains, part I, Theoretical Computer Science 13 (1) (1981) 85–108.
 - [15] I. Castellani, M. Dezani-Ciancaglini, P. Giannini, [Event structure semantics for multiparty sessions](#), Journal of Logical and Algebraic Methods in Programming 131 (2023) 100844. doi:<https://doi.org/10.1016/j.jlamp.2022.100844>.
URL <https://www.sciencedirect.com/science/article/pii/S2352220822000979>
 - [16] L. Edixhoven, S. Jongmans, J. Proença, G. Cledou, [Branching pomsets for choreographies](#), in: C. Aubert, C. D. Giusto, L. Safina, A. Scalas (Eds.), Proceedings 15th Interaction and Concurrency Experience, ICE 2022, Lucca, Italy, 17th June 2022, Vol. 365 of EPTCS, 2022, pp. 37–52. doi:[10.4204/EPTCS.365.3](https://doi.org/10.4204/EPTCS.365.3).
URL <https://doi.org/10.4204/EPTCS.365.3>
 - [17] L. Edixhoven, S. Jongmans, [Realisability of branching pomsets](#), in: S. L. T. Tarifa, J. Proença (Eds.), Formal Aspects of Component Software - 18th International Conference, FACS 2022, Virtual Event, November 10-11, 2022, Proceedings, Vol. 13712 of Lecture Notes in Computer Science, Springer, 2022, pp. 185–204. doi:[10.1007/978-3-031-20872-0_11](https://doi.org/10.1007/978-3-031-20872-0_11).
URL https://doi.org/10.1007/978-3-031-20872-0_11
 - [18] V. R. Pratt, [Modeling concurrency with partial orders](#), Int. J. Parallel Program. 15 (1) (1986) 33–71. doi:[10.1007/BF01379149](https://doi.org/10.1007/BF01379149).
URL <https://doi.org/10.1007/BF01379149>
 - [19] G. Winskel, Events in computation, Ph.D. thesis, University of Edinburgh (1980).

- [20] Y. Arbach, D. S. Karcher, K. Peters, U. Nestmann, [Dynamic causality in event structures](#), Log. Methods Comput. Sci. 14 (1) (2018). doi: [10.23638/LMCS-14\(1:17\)2018](#).
URL [https://doi.org/10.23638/LMCS-14\(1:17\)2018](https://doi.org/10.23638/LMCS-14(1:17)2018)
- [21] G. Winskel, Event structure semantics for CCS and related languages, in: M. Nielsen, E. M. Schmidt (Eds.), ICALP, Vol. 140 of LNCS, Springer, Heidelberg, 1982, pp. 561–576.
- [22] R. Langerak, Bundle event structures: a non-interleaving semantics for LOTOS, in: M. Diaz, R. Groz (Eds.), Formal Description Techniques for Distributed Systems and Communication Protocols, North-Holland, Amsterdam, 1993, pp. 331–346.
- [23] G. Boudol, I. Castellani, Permutation of transitions: an event structure semantics for CCS and SCCS, in: J. W. de Bakker, W. P. de Roever, G. Rozenberg (Eds.), REX: Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, Vol. 354 of LNCS, Springer, Heidelberg, 1988, pp. 411–427.
- [24] G. Boudol, I. Castellani, Flow models of distributed computations: event structures and nets, Research Report 1482, INRIA (1991).
- [25] P. Baldan, A. Corradini, U. Montanari, [Contextual petri nets, asymmetric event structures, and processes](#), Information and Computation 171 (1) (2001) 1–49. doi:<https://doi.org/10.1006/inco.2001.3060>.
URL <https://www.sciencedirect.com/science/article/pii/S0890540101930603>
- [26] R. Langerak, Transformations and semantics for LOTOS, 1992.
- [27] G. Boudol, I. Castellani, On the semantics of concurrency: partial orders and transition systems, in: H. Ehrig, R. A. Kowalski, G. Levi, U. Montanari (Eds.), TAPSOFT, Vol. 249 of LNCS, Springer, Heidelberg, 1987, pp. 123–137.
- [28] R. J. van Glabbeek, G. D. Plotkin, [Event structures for resolvable conflict](#), in: J. Fiala, V. Koubek, J. Kratochvíl (Eds.), Mathematical Foundations of Computer Science 2004, 29th International Symposium, MFCS 2004, Prague, Czech Republic, August 22–27, 2004, Proceedings, Vol.

- 3153 of Lecture Notes in Computer Science, Springer, 2004, pp. 550–561. [doi:10.1007/978-3-540-28629-5_42](https://doi.org/10.1007/978-3-540-28629-5_42).
URL https://doi.org/10.1007/978-3-540-28629-5_42
- [29] J. Katoen, L. Lambert, Pomsets for MSC, in: H. König, P. Langendörfer (Eds.), *Formale Beschreibungstechniken für verteilte Systeme*, 8. GI/ITG-Fachgespräch, Cottbus, 4. und 5. Juni 1998, Verlag Shaker, 1998, pp. 197–207.
 - [30] A. Rensink, H. Wehrheim, Process algebra with action dependencies, *Acta Informatica* 38 (3) (2001) 155–234. [doi:10.1007/s002360100070](https://doi.org/10.1007/s002360100070).
 - [31] D. Sangiorgi, *Introduction to bisimulation and coinduction*, Cambridge University Press, 2011. [doi:10.1017/CB09780511777110](https://doi.org/10.1017/CB09780511777110).
 - [32] L. Edixhoven, S.-S. Jongmans, J. Proença, G. Cledou, *Branching pomsets for choreographies (technical report)*, Tech. Rep. OUNL-CS-2022-04, Open University of the Netherlands (2022).
 - [33] L. Edixhoven, S.-S. Jongmans, *Realisability of branching pomsets (technical report)*, Tech. Rep. OUNL-CS-2022-05, Open University of the Netherlands (2022).
 - [34] D. Brand, P. Zafiropulo, *On communicating finite-state machines*, *J. ACM* 30 (2) (1983) 323–342. [doi:10.1145/322374.322380](https://doi.org/10.1145/322374.322380).
URL <https://doi.org/10.1145/322374.322380>
 - [35] L. P. Cordella, P. Foggia, C. Sansone, M. Vento, *An efficient algorithm for the inexact matching of ARG graphs using a contextual transformational model*, in: *13th International Conference on Pattern Recognition, ICPR 1996, Vienna, Austria, 25-19 August, 1996*, IEEE Computer Society, 1996, pp. 180–184. [doi:10.1109/ICPR.1996.546934](https://doi.org/10.1109/ICPR.1996.546934).
URL <https://doi.org/10.1109/ICPR.1996.546934>
 - [36] E. Tuosto, R. Guanciale, *Semantics of global view of choreographies*, *J. Log. Algebraic Methods Program.* 95 (2018) 17–40. [doi:10.1016/j.jlamp.2017.11.002](https://doi.org/10.1016/j.jlamp.2017.11.002).
 - [37] R. Alur, K. Etessami, M. Yannakakis, *Inference of message sequence charts*, *IEEE Trans. Software Eng.* 29 (7) (2003) 623–633. [doi:10.1109/32.1192000](https://doi.org/10.1109/32.1192000).

1109/TSE.2003.1214326.

URL <https://doi.org/10.1109/TSE.2003.1214326>

- [38] A. Finkel, É. Lozes, [Synchronizability of communicating finite state machines is not decidable](#), in: I. Chatzigiannakis, P. Indyk, F. Kuhn, A. Muscholl (Eds.), 44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland, Vol. 80 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, pp. 122:1–122:14. [doi:10.4230/LIPIcs.ICALP.2017.122](#).
URL <https://doi.org/10.4230/LIPIcs.ICALP.2017.122>
- [39] R. Milner, [A Calculus of Communicating Systems](#), Vol. 92 of Lecture Notes in Computer Science, Springer, 1980. [doi:10.1007/3-540-10235-3](#).
URL <https://doi.org/10.1007/3-540-10235-3>
- [40] D. Sangiorgi, D. Walker, [The Pi-Calculus - a theory of mobile processes](#), Cambridge University Press, 2001.
- [41] M. Carbone, N. Yoshida, K. Honda, [Asynchronous session types: Exceptions and multiparty interactions](#), in: M. Bernardo, L. Padovani, G. Zavattaro (Eds.), Formal Methods for Web Services, 9th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2009, Bertinoro, Italy, June 1-6, 2009, Advanced Lectures, Vol. 5569 of Lecture Notes in Computer Science, Springer, 2009, pp. 187–212. [doi:10.1007/978-3-642-01918-0_5](#).
URL https://doi.org/10.1007/978-3-642-01918-0_5
- [42] P. Deniélou, N. Yoshida, A. Bejleri, R. Hu, [Parameterised multiparty session types](#), Log. Methods Comput. Sci. 8 (4) (2012). [doi:10.2168/LMCS-8\(4:6\)2012](#).
URL [https://doi.org/10.2168/LMCS-8\(4:6\)2012](https://doi.org/10.2168/LMCS-8(4:6)2012)
- [43] A. Scalas, N. Yoshida, [Less is more: multiparty session types revisited](#), Proc. ACM Program. Lang. 3 (POPL) (2019) 30:1–30:29. [doi:10.1145/3290343](#).
URL <https://doi.org/10.1145/3290343>
- [44] G. Castagna, M. Dezani-Ciancaglini, L. Padovani, [On global types and multi-party session](#), Log. Methods Comput. Sci. 8 (1) (2012). [doi:](#)

10.2168/LMCS-8(1:24)2012.

URL [https://doi.org/10.2168/LMCS-8\(1:24\)2012](https://doi.org/10.2168/LMCS-8(1:24)2012)

- [45] L. Bocchi, H. C. Melgratti, E. Tuosto, [Resolving non-determinism in choreographies](#), in: Z. Shao (Ed.), Programming Languages and Systems - 23rd European Symposium on Programming, ESOP 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings, Vol. 8410 of Lecture Notes in Computer Science, Springer, 2014, pp. 493–512. doi:10.1007/978-3-642-54833-8_26.
URL https://doi.org/10.1007/978-3-642-54833-8_26
- [46] I. Castellani, M. Dezani-Ciancaglini, P. Giannini, [Reversible sessions with flexible choices](#), Acta Informatica 56 (7-8) (2019) 553–583. doi:10.1007/s00236-019-00332-y.
URL <https://doi.org/10.1007/s00236-019-00332-y>
- [47] S. Jongmans, N. Yoshida, [Exploring type-level bisimilarity towards more expressive multiparty session types](#), in: P. Müller (Ed.), Programming Languages and Systems - 29th European Symposium on Programming, ESOP 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings, Vol. 12075 of Lecture Notes in Computer Science, Springer, 2020, pp. 251–279. doi:10.1007/978-3-030-44914-8_10.
URL https://doi.org/10.1007/978-3-030-44914-8_10
- [48] I. Castellani, G. Zhang, [Parallel product of event structures](#), Theor. Comput. Sci. 179 (1-2) (1997) 203–215. doi:10.1016/S0304-3975(96)00104-1.
URL [https://doi.org/10.1016/S0304-3975\(96\)00104-1](https://doi.org/10.1016/S0304-3975(96)00104-1)
- [49] T. Kappé, P. Brunet, B. Luttik, A. Silva, F. Zanasi, [On series-parallel pomset languages: Rationality, context-freeness and automata](#), J. Log. Algebraic Methods Program. 103 (2019) 130–153. doi:10.1016/j.jlamp.2018.12.001.
URL <https://doi.org/10.1016/j.jlamp.2018.12.001>
- [50] M. Hennessy, R. Milner, [Algebraic laws for nondeterminism and concurrency](#), J. ACM 32 (1) (1985) 137–161. doi:10.1145/2455.2460.
URL <https://doi.org/10.1145/2455.2460>

- [51] G. L. Ferrari, R. Gorrieri, U. Montanari, [An extended expansion theorem](#), in: S. Abramsky, T. S. E. Maibaum (Eds.), TAPSOFT'91: Proceedings of the International Joint Conference on Theory and Practice of Software Development, Brighton, UK, April 8-12, 1991, Volume 2: Advances in Distributed Computing (ADC) and Colloquium on Combining Paradigms for Software Development (CCPSD), Vol. 494 of Lecture Notes in Computer Science, Springer, 1991, pp. 29–48. [doi: 10.1007/3540539816_56](#).
URL https://doi.org/10.1007/3540539816_56