Branching pomsets for choreographies

Luc Edixhoven   Sung-Shik Jongmans
Open University (Heerlen) and
CWI (Amsterdam), Netherlands
{led,ssj}@ou.nl

José Proença
CISTER, ISEP,
Polytechnic Institute of Porto, Portugal
pro@isep.ipp.pt

Guillermina Cledou
HASLab, INESC TEC & University of Minho, Portugal
mgc@inesctec.pt

Choreographic languages describe possible sequences of interactions among a set of agents. Typical denotational models are based on languages or automata over sending and receiving actions. Pomsets provide a more compact alternative by using a partial order over these actions and by not making explicit the possible interleaving of concurrent actions. However, pomsets offer no compact representation of choices. 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 encode choreographies as branching pomsets and show that the pomset semantics of the encoded choreographies are bisimilar to their operational semantics.

1 Introduction

Choreographic languages describe possible sequences of interactions, or communication protocols, among a set of agents. Their use is well established [9, 1, 7, 8, 2, 5], and it typically includes (1) reasoning statically over interaction properties and (2) generating code that facilitates the implementation of the concurrent protocols. Static properties include deadlock absence or the equivalence between global protocols and the parallel composition of local protocols for each agent. The code generated from choreographic languages include skeleton code for concurrent code, generated behavioural types that can be used to type-check agents, or dedicated orchestrators that dictate how the agents can interact. In this work we focus on how to analyse choreographies by proposing a new structure to compactly represent their behaviour, based on *partial-ordered multisets* (pomsets). We foresee applications of this work in both aforementioned areas.

We use two simple running examples to motivate our approach.

1. **Master-workers (MW) protocol [11].** A *master* (m) concurrently sends *tasks* ($\mathtt{t}$) to some number of *workers* ($\mathsf{w}_1, \dots, \mathsf{w}_n$). Once workers finish their task, they inform the master that they are *done* ($\mathtt{d}$). This protocol is expressed in our choreographic language as follows for the case of two workers.

$$(\mathsf{m}{\rightarrow}\mathsf{w}_1{:}\mathtt{t}\,;\mathsf{w}_1{\rightarrow}\mathsf{m}{:}\mathtt{d}) \ \| \ (\mathsf{m}{\rightarrow}\mathsf{w}_2{:}\mathtt{t}\,;\mathsf{w}_2{\rightarrow}\mathsf{m}{:}\mathtt{d}).$$

Here, $\mathsf{m}{\rightarrow}\mathsf{w}_1{:}\mathtt{t}$ represents an asynchronous communication from m to w_1 of a message of type $\mathtt{t}$, ';' represents sequential composition and '$\|$' represents parallel composition.

Submitted to:
ICE 2022

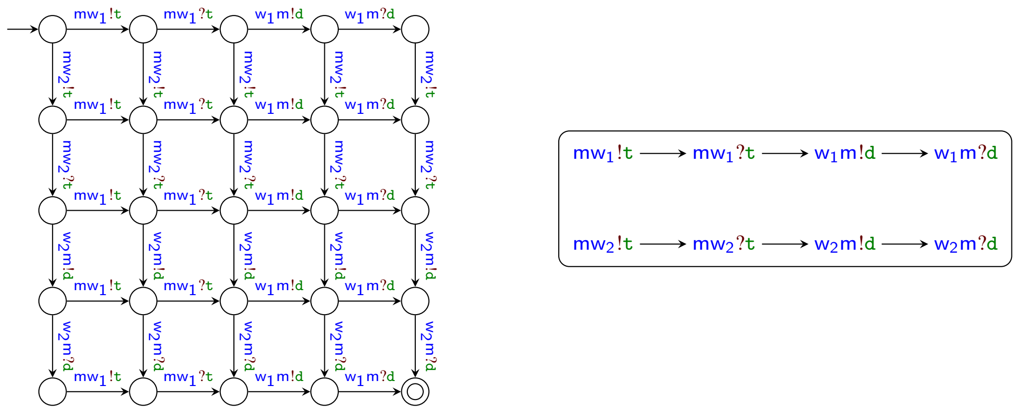


Figure 1: An automaton (left) and a pomset (right) representing the master-workers protocol.

2. **Distributed voting (DV) protocol.** Three participants – Alice (a), Bob (b) and Carol (c) – send their vote (*yes* (y) or *no* (n)) to every other participant in parallel. This is expressed as follows, where + indicates nondeterministic choice.

$$\begin{pmatrix}(a{\to}b{:}y \parallel a{\to}c{:}y)+\\ (a{\to}b{:}n \parallel a{\to}c{:}n)\end{pmatrix} \parallel \begin{pmatrix}(b{\to}a{:}y \parallel b{\to}c{:}y)+\\ (b{\to}a{:}n \parallel b{\to}c{:}n)\end{pmatrix} \parallel \begin{pmatrix}(c{\to}a{:}y \parallel c{\to}b{:}y)+\\ (c{\to}a{:}n \parallel c{\to}b{:}n)\end{pmatrix}$$

A protocol can evolve by performing sequences of sending and receiving actions. E.g., $ab!x$ denotes a sending action from a to b with a message of type x, and $ab?x$ denotes the dual receiving action on b. Protocols with parallel interactions can have an explosion of states, such as our MW protocol, whose full state machine can be found on the left of Figure 1. To avoid this explosion, the state space can be represented more compactly using so-called *partially ordered multisets*, or simply pomsets [10, 6]. The right of Figure 1 shows a graphical pomset representation of the same MW protocol. The pomset contains eight events, whose labels are shown. The arrows visualise the partial order: an event precedes any other event to which it has an outgoing arrow, either directly or transitively. In this example, the event with label $mw_1!t$ precedes the event with label $mw_1?t$ directly and the events with labels $w_1m!d$ and $w_1m?d$ transitively. However, it is independent of the events involving w_2.

The behaviour represented by a pomset is the set of all its linearisations, i.e., all sequences of the labels of its events that respect their partial order. The set of linearisations of the pomset in Figure 1 consists of all interleavings of the two threads $mw_1!t\,mw_1?t\,w_1m!d\,w_1m?d$ and $mw_2!t\,mw_2?t\,w_2m!d\,w_2m?d$. This explicit concurrency yields a compact representation of the possible interleavings using just $4+4$ events, whereas the state machine needs 5×5 states to represent all interleavings. If we were to add a third worker, the automaton would grow by another factor 5, while the pomset would expand by just four additional events.

While pomsets can compactly represent concurrent behaviour, **choices** need to be represented as *sets* of pomsets: one for every branch. As a consequence, one might need an exponential number of pomsets to represent a protocol with many choices. The exponential growth is visible in our DV protocol with three participants, depicted on the left side of Figure 2. This diagram represents a set of pomsets that capture the protocol's possible behaviour, counting $2\times 2\times 2$ different pomsets. If we were to add a fourth participant, the set would grow by another factor 2.

This paper proposes an extension to pomsets, named *branching pomsets*, with a branching structure that can compactly represent choices. A branching pomset initially contains all branches of choices,

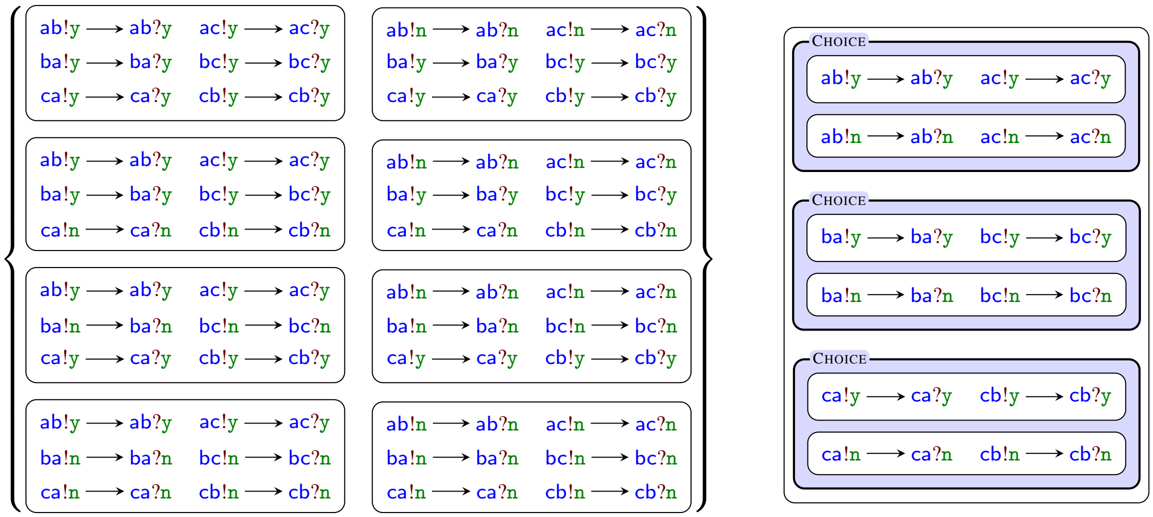


Figure 2: A set of pomsets (left) and a branching pomset (right) representing a three-participant distributed vote.

and discards non-chosen branches when firing events that require resolving a choice. The right side of Figure 2 depicts an example of a branching pomset for our DV protocol: where we would traditionally need $2 \times 2 \times 2$ pomsets (with six pairs of events each), we can represent the same behaviour as a single branching pomset with $2+2+2$ choices (with four pairs of events each). Adding an additional participant would double the number of pomsets in the set of pomsets, while it would add a single choice to the branching pomset.

To aid in the understanding of branching pomsets and their semantics, we provide a prototype tool to visualise them, available at `https://arca.di.uminho.pt/b-pomset/`. The tool provides a web interface where one can submit an input choreography, which is then visualised as a branching pomset and can be simulated. The examples and figures in the paper are already available as preset inputs. We note that the pomset simulation in our prototype currently does not support loops, for reasons which will become apparent later in the paper; however, all other operators are supported and we are most interested in (combinations of) choice and parallel composition.

Contribution This paper provides three core contributions: (1) an extension of pomsets with a branching structure, named branching pomsets, (2) an encoding from a choreographic language into branching pomsets, and (3) a formal proof that the operational semantics of a choreography and of its encoded branching pomset are equivalent, i.e., bisimilar.

Structure of the paper Section 2 presents the syntax of our choreography language and its operational semantics. Section 3 formalises branching pomsets and their semantics. Section 4 formalises how to obtain a branching pomset from a choreography and shows that a choreography and its derived branching pomset are behaviourally equivalent. Finally, Section 5 presents our conclusions and a brief discussion about future work and related work.

$$c ::= \mathbf{0} \mid \mathsf{a}{\to}\mathsf{b}{:}\mathtt{x} \mid \boxed{\mathsf{ab}?\mathtt{x}} \mid c\,;c \mid c+c \mid c \parallel c \mid c^*$$

Figure 3: Syntax of choreographies, where a and b are participants ($\mathsf{a} \neq \mathsf{b}$) and $\mathtt{x}$ is a message type.

2 Choreographies

In this section, we formally define the syntax and semantics of our choreographic language, examples of which have been shown in the previous section.

Let $\mathcal{A}$ be the set of all participants $\mathsf{a}, \mathsf{b}, \ldots$. Let $\mathcal{L}$ be the set of actions $\{\mathsf{ab}!\mathtt{x}, \mathsf{ab}?\mathtt{x}\}$ for all participants $\mathsf{a} \neq \mathsf{b}$ and message types $\mathtt{x}$. For all actions the *subject* of the action is its active participant: the subject of a send action $\mathsf{ab}!\mathtt{x}$ is a and the subject of a receive action $\mathsf{ab}?\mathtt{x}$ is b.

The syntax is formally defined in Figure 3. Its components are standard: '$\mathbf{0}$' is the empty choreography; '$\mathsf{a}{\to}\mathsf{b}{:}\mathtt{x}$' is the asynchronous communication from a to b of a message of type $\mathtt{x}$; the boxed term '$\mathsf{ab}?\mathtt{x}$' represents a pending receive on b from a of a message of type $\mathtt{x}$, it is boxed in Figure 3 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. More traditionally, 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 $\mathsf{a}{\to}\mathsf{b}{:}\mathtt{x}\,;\mathsf{c}{\to}\mathsf{d}{:}\mathtt{x}$ we can execute the action $\mathsf{cd}!\mathtt{x}$ as it does not affect the participants of $\mathsf{a}{\to}\mathsf{b}{:}\mathtt{x}$: there is no dependency and thus no need to wait for $\mathsf{a}{\to}\mathsf{b}{:}\mathtt{x}$ to go first. However, in $\mathsf{a}{\to}\mathsf{b}{:}\mathtt{x}\,;\mathsf{a}{\to}\mathsf{c}{:}\mathtt{x}$ the action $\mathsf{ac}!\mathtt{x}$ cannot be executed first as its subject (a) must first execute $\mathsf{ab}!\mathtt{x}$. This is the common interpretation of sequential composition in the context of message sequence charts [10], multiparty session types [7] and choreographic programming [2].

The reduction rules of our choreographic language are formally defined in Figure 4a and its termination rules in Figure 4b. To formalise the reduction of weak sequential composition, we follow Rensink and Wehrheim [13], who define a notion of *partial termination*.

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 the subject of ℓ. Conceptually, a choreography c_1 can partially terminate for the subject of 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 $c_1 \xrightarrow{\checkmark_\ell} c_1'$, where c_1' is the remainder of c_1 after discarding all branches involving the subject of ℓ. For example, if $c_1 = \mathsf{a}{\to}\mathsf{b}{:}\mathtt{x} + \mathsf{a}{\to}\mathsf{c}{:}\mathtt{x}$ then $c_1 \xrightarrow{\checkmark_{\mathsf{cd}!\mathtt{x}}} \mathsf{a}{\to}\mathsf{b}{:}\mathtt{x}$, as this branch does not contain 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 the subject of ℓ, i.e., $c_1 \not\xrightarrow{\checkmark_\ell}$. In the above example, $c_1 \not\xrightarrow{\checkmark_{\mathsf{ad}!\mathtt{x}}}$.

The rules for partial termination are deterministic and only discard the absolutely necessary. In the example above, $c_1 \xrightarrow{\checkmark_{\mathsf{da}!\mathtt{x}}} c_1$ 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 4c. We highlight the rules for 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.
- A choice $c_1 + c_2$ can partially terminate if at least one of its branches can. If both branches can

$$\frac{}{\mathsf{a}{\rightarrow}\mathsf{b}{:}\mathtt{x} \xrightarrow{\mathsf{ab}!\mathtt{x}} \mathsf{ab}?\mathtt{x}} \qquad \frac{}{\mathsf{ab}?\mathtt{x} \xrightarrow{\mathsf{ab}?\mathtt{x}} \mathbf{0}} \qquad \frac{c_1 \xrightarrow{\ell} c_1'}{c_1 \,;\, c_2 \xrightarrow{\ell} c_1' \,;\, c_2} \qquad \frac{c_1 \xrightarrow{\checkmark_\ell} c_1' \quad c_2 \xrightarrow{\ell} c_2'}{c_1 \,;\, c_2 \xrightarrow{\ell} c_1' \,;\, c_2'}$$

$$\frac{c_1 \xrightarrow{\ell} c_1'}{c_1 \parallel c_2 \xrightarrow{\ell} c_1' \parallel c_2} \qquad \frac{c_2 \xrightarrow{\ell} c_2'}{c_1 \parallel c_2 \xrightarrow{\ell} c_1 \parallel c_2'} \qquad \frac{c_1 \xrightarrow{\ell} c_1'}{c_1 + c_2 \xrightarrow{\ell} c_1'} \qquad \frac{c_2 \xrightarrow{\ell} c_2'}{c_1 + c_2 \xrightarrow{\ell} c_2'} \qquad \frac{c \xrightarrow{\ell} c'}{c^* \xrightarrow{\ell} c' \,;\, c^*}$$

(a) Reduction rules.

$$\frac{}{\mathbf{0}\downarrow} \qquad \frac{}{c^*\downarrow} \qquad \frac{c_1\downarrow \quad c_2\downarrow \quad \dagger \in \{;, \parallel\}}{c_1 \dagger c_2\downarrow} \qquad \frac{c_i\downarrow \quad i \in \{1,2\}}{c_1 + c_2\downarrow}$$

(b) Termination rules.

$$\frac{}{\mathbf{0} \xrightarrow{\checkmark_\ell} \mathbf{0}} \qquad \frac{c \xrightarrow{\checkmark_\ell} c}{c^* \xrightarrow{\checkmark_\ell} c^*} \qquad \frac{c \not\xrightarrow{\checkmark_\ell} c}{c^* \xrightarrow{\checkmark_\ell} \mathbf{0}} \qquad \frac{c_1 \xrightarrow{\checkmark_\ell} c_1' \quad c_2 \xrightarrow{\checkmark_\ell} c_2' \quad \dagger \in \{;, \parallel, +\}}{c_1 \dagger c_2 \xrightarrow{\checkmark_\ell} c_1' \dagger c_2'}$$

$$\frac{c_1 \xrightarrow{\checkmark_\ell} c_1' \quad c_2 \not\xrightarrow{\checkmark_\ell}}{c_1 + c_2 \xrightarrow{\checkmark_\ell} c_1'} \qquad \frac{c_1 \not\xrightarrow{\checkmark_\ell} \quad c_2 \xrightarrow{\checkmark_\ell} c_2'}{c_1 + c_2 \xrightarrow{\checkmark_\ell} c_2'} \qquad \frac{\mathit{subj}(\ell) \notin \{\mathsf{a}, \mathsf{b}\}}{\mathsf{a}{\rightarrow}\mathsf{b}{:}\mathtt{x} \xrightarrow{\checkmark_\ell} \mathsf{a}{\rightarrow}\mathsf{b}{:}\mathtt{x}} \qquad \frac{\mathit{subj}(\ell) \neq \mathsf{b}}{\mathsf{ab}?\mathtt{x} \xrightarrow{\checkmark_\ell} \mathsf{ab}?\mathtt{x}}$$

(c) Partial termination rules.

Figure 4: Operational semantics of choreographies.

partially terminate then both are kept, otherwise only the partially terminated one is kept.

- Following Rensink and Wehrheim, a loop c^* can partially terminate if its body (c) can partially terminate without discarding any branches, i.e., if $c \xrightarrow{\checkmark_\ell} c$. In that case also $c^* \xrightarrow{\checkmark_\ell} c^*$. Otherwise we allow c^* to be skipped entirely, represented as partial termination to $\mathbf{0}$, i.e., $c^* \xrightarrow{\checkmark_\ell} \mathbf{0}$. 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 $c \not\xrightarrow{\checkmark_\ell} c$ as a shorthand to cover both these cases. Skipping a loop is necessary, for example, in a modified master-workers protocol where the master can send an arbitrary number of tasks to the workers, followed by an `end` message to indicate termination. With one worker, this protocol is expressed as $(\mathsf{m}{\rightarrow}\mathsf{w}_1{:}\mathtt{t} \,;\, \mathsf{w}_1{\rightarrow}\mathsf{m}{:}\mathtt{d})^* \,;\, \mathsf{m}{\rightarrow}\mathsf{w}_1{:}\mathtt{end}$. In this choreography, the loop has to eventually partially terminate to $\mathbf{0}$ to allow for the action $\mathsf{mw}_1!\mathtt{end}$.

Example 1. Let $c_1 = (\mathsf{a}{\rightarrow}\mathsf{b}{:}\mathtt{x} + \mathsf{a}{\rightarrow}\mathsf{c}{:}\mathtt{x}) \,;\, (\mathsf{d}{\rightarrow}\mathsf{b}{:}\mathtt{x} + \mathsf{d}{\rightarrow}\mathsf{e}{:}\mathtt{x})$. Let $c_2 = (\mathsf{a}{\rightarrow}\mathsf{b}{:}\mathtt{x} + \mathsf{c}{\rightarrow}\mathsf{b}{:}\mathtt{x})^* \parallel (\mathsf{c}{\rightarrow}\mathsf{a}{:}\mathtt{x} + \mathsf{c}{\rightarrow}\mathsf{b}{:}\mathtt{x})$.

- $c_1 \xrightarrow{\checkmark_{\mathsf{be}!\mathtt{x}}} \mathsf{a}{\rightarrow}\mathsf{c}{:}\mathtt{x} \,;\, \mathsf{d}{\rightarrow}\mathsf{e}{:}\mathtt{x}$. The subject b of $\mathsf{be}!\mathtt{x}$ occurs in one branch of each of both choices. While the recipient e also occurs in the second branch of the second choice, since it is not the actual subject it does not create a conflict.
- $c_1 \not\xrightarrow{\checkmark_{\mathsf{ab}!\mathtt{x}}}$. While the second choice can partially terminate without reducing, the first choice contains the subject a of $\mathsf{ab}!\mathtt{x}$ in both of its branches. Since one of the choices cannot reduce, neither can their sequential composition.
- $c_2 \xrightarrow{\checkmark_{\mathsf{ad}!\mathtt{x}}} \mathbf{0} \parallel \mathsf{c}{\rightarrow}\mathsf{b}{:}\mathtt{x}$. The subject a of $\mathsf{ad}!\mathtt{x}$ only occurs in one branch of the loop body, but the loop

can only reduce to $\mathbf{0}$. On the right hand side of the parallel composition, a occurs only in the first branch.

- $c_2 \not\xrightarrow{\checkmark \mathsf{cd!x}}$. While the loop can again reduce to $\mathbf{0}$, the subject c of $\mathsf{cd!x}$ occurs in both branches of the right hand side of the parallel composition. Since its right hand side cannot partially terminate, neither can it as a whole.

As already discovered by Rensink and Wehrheim [13], an unwanted consequence of these rules for partial termination is that unfolding iterations of loops no longer preserves behaviour. We would like c^* and $(c\,;c^*)+\mathbf{0}$ to behave the same, but this is not the case. For example, if $c = \mathsf{a{\to}b{:}x} + \mathsf{c{\to}d{:}x}$, then $c^* \xrightarrow{\checkmark \mathsf{ab!x}} \mathbf{0}$ but $(c\,;c^*)+\mathbf{0} \xrightarrow{\checkmark \mathsf{ab!x}} (\mathsf{c{\to}d{:}x}\,;\mathbf{0})+\mathbf{0}$. Then $c^*\,;c \xrightarrow{\mathsf{ab!x}} \mathsf{ab?x}$ by skipping the loop; however, $((c\,;c^*)+\mathbf{0})\,;c$ has no way to match this as it can skip the loop but it can only reduce the already unfolded iteration c to $\mathsf{c{\to}d{:}x}$ — 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 the subject of 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 the subject of ℓ without having to reduce, or it cannot partially terminate at all. Formally: if $c \xrightarrow{\checkmark_\ell} c'$ 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 $c^* \xrightarrow{\checkmark_\ell} \mathbf{0}$ then $c \not\xrightarrow{\checkmark_\ell}$ and $(c\,;c^*)+\mathbf{0}$ is also forced to reduce to the second branch of the choice, which is $\mathbf{0}$. More precisely, let c^* be some dependently guarded expression. If $c \xrightarrow{\checkmark_\ell} c'$ for some ℓ, c', then $c' = c$. It follows that $c^* \xrightarrow{\checkmark_\ell} c^*$ and $(c\,;c^*)+\mathbf{0} \xrightarrow{\checkmark_\ell} (c\,;c^*)+\mathbf{0}$. Similarly, if $c \not\xrightarrow{\checkmark_\ell}$ then $c^* \xrightarrow{\checkmark_\ell} \mathbf{0}$ and $(c\,;c^*)+\mathbf{0} \xrightarrow{\checkmark_\ell} \mathbf{0}$.

Example 2. Let $c_1 = \mathsf{a{\to}b{:}x} + \mathsf{a{\to}c{:}x}$. Let $c_2 = \mathsf{a{\to}b{:}x} + \mathsf{b{\to}a{:}x}$.

- c_1^* is not dependently guarded as $c_1 \xrightarrow{\checkmark \mathsf{cd!x}} \mathsf{a{\to}b{:}x} \neq c_1$. However, c_1 itself *is* dependently guarded as it does not contain any loop.
- c_2^* is dependently guarded since both a and b occur in both branches of c_2. However, $(c_2^*)^*$ is *not* dependently guarded, since $c_2^* \xrightarrow{\checkmark \mathsf{ab!x}} \mathbf{0}$.

3 Branching pomsets

In this section, we formally define the syntax and semantics of branching pomsets. Additionally, we define a pomset interpretation of expressions in our choreographic language and we show this interpretation to be faithful by showing that it is bisimilar to the original choreography.

A partially ordered multiset [12], or pomset for short, consist of a set of nodes E (events), a labelling function λ to map events to some set of labels (e.g., send and receive actions), and a partial order $\leq$ to define dependencies between pairs of events (e.g., an event, or rather its corresponding action, can only fire if all events preceding it in the partial order have already fired). Its behaviour is the set of all sequences of the labels of its events that abide by $\leq$.

For example, for the pomset in Figure 1, $E = \{e_1,\ldots,e_8\}$, $\lambda = \{e_1 \mapsto \mathsf{mw_1!t}, e_2 \mapsto \mathsf{mw_1?t}, e_3 \mapsto \mathsf{w_1m!d}, e_4 \mapsto \mathsf{w_1m?d}, e_5 \mapsto \mathsf{mw_2!t}, e_6 \mapsto \mathsf{mw_2?t}, e_7 \mapsto \mathsf{w_2m!d}, e_8 \mapsto \mathsf{w_2m?d}\}$, and $\leq\, = \{(e_i,e_j) \mid (i,j \in [1,4] \vee i,j \in [5,8]) \wedge i \leq j\}$. Its behaviour consists of all interleavings of $\mathsf{mw_1!tmw_1?tw_1m!dw_1m?d}$ and $\mathsf{mw_2!tmw_2?tw_2m!dw_2m?d}$.

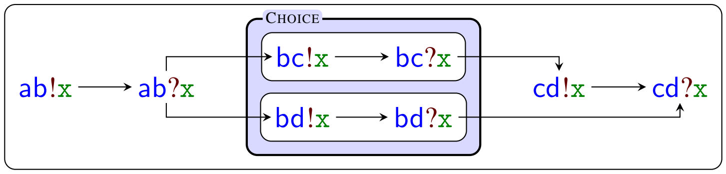


Figure 5: A branching pomset representing the choreography a→b:x ; (b→c:x + b→d:x) ; c→d:x.

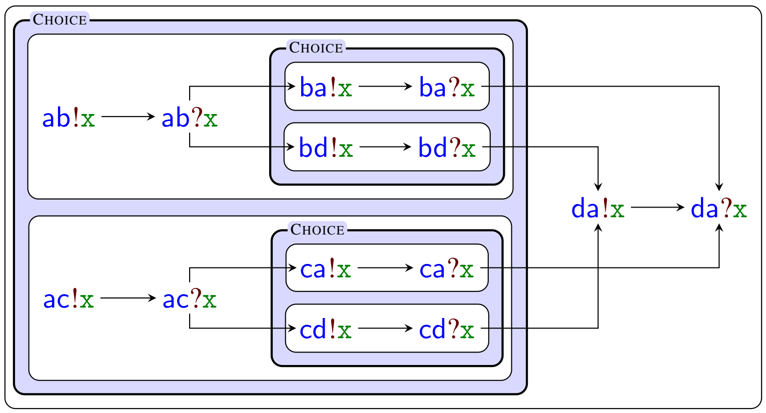


Figure 6: A branching pomset representing the choreography ((a→b:x ; (b→a:x + b→d:x)) + (a→c:x ; (c→a:x + c→d:x))) ; d→a:x.

As illustrated in Figure 2, however, traditional pomsets suffer from the same problem when representing choices that automata suffer from when representing concurrency: there is no explicit representation of choices in pomsets, and they are represented only implicitly by using a set of possible pomsets. We tackle this by extending pomsets with an explicit representation of choices: a branching structure on events.

Branching structure The general idea is of a branching pomset is that all possible events are initially part of the pomset, but that some are defined as being part of some choice. To fire these, all relevant choices must first be resolved by replacing the choice with one of its branches, thereby discarding the other branch. This same idea governs the operational semantics of choreographies (Figure 4): both branches of a choice are initially part of the choreography but, to proceed in one of them, the other must be dropped.

The branching structure does not interrupt the partial order and all events still participate in it, as shown in Figure 5, where arrows flow both into and out of the branches of the choice. As such, a choice can also be resolved to fire an event which is only preceded by one of the branches, reminiscent of the partial termination of choices (Figure 4c). For example, in Figure 5 the upper branch (b→c:x) can be discarded to fire the event labelled cd!x, as it is not dependent on the lower branch. As shown in Figure 6, nested choices are supported as well.

Formally, the branching structure is defined as below as a tree with root node $\mathcal{B}$, whose children are either a single event e or a choice node $\mathcal{C}$ with children (branches) $\mathcal{B}_1, \mathcal{B}_2$. All leaves are events.

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

For example, for the pomset in Figure 5, if $E = \{e_1, \dots, e_8\}$ and $\lambda = \{e_1 \mapsto \mathtt{ab!x}, e_2 \mapsto \mathtt{ab?x}, e_3 \mapsto \mathtt{cd!x}, e_4 \mapsto \mathtt{cd?x}, e_5 \mapsto \mathtt{bc!x}, e_6 \mapsto \mathtt{bc?x}, e_7 \mapsto \mathtt{bd!x}, e_8 \mapsto \mathtt{bd?x}\}$, then its branching structure is $\{e_1, e_2, e_3, e_4, \{\{e_5, e_6\}, \{e_7, e_8\}\}\}$. For the pomset in Figure 6, if $E = \{e_1, \dots, e_{14}\}$ and $\lambda = \{e_1 \mapsto \mathtt{ab!x}, e_2 \mapsto \mathtt{ab?x}, e_3 \mapsto \mathtt{ba!x}, e_4 \mapsto \mathtt{ba?x}, e_5 \mapsto \mathtt{bd!x}, e_6 \mapsto \mathtt{bd?x}, e_7 \mapsto \mathtt{ac!x}, e_8 \mapsto \mathtt{ac?x}, e_9 \mapsto \mathtt{ca!x}, e_{10} \mapsto \mathtt{ca?x}, e_{11} \mapsto \mathtt{cd!x}, e_{12} \mapsto \mathtt{cd?x}, e_{13} \mapsto \mathtt{da!x}, e_{14} \mapsto \mathtt{da?x}\}$, then its branching structure is $\{e_{13}, e_{14}, \{\{e_1, e_2, \{\{e_3, e_4\}, \{e_5, e_6\}\}\}, \{e_7, e_8, \{\{e_9, e_{10}\}, \{e_{11}, e_{12}\}\}\}\}\}$. By resolving the outer choice and picking its upper branch ($\mathtt{a}{\rightarrow}\mathtt{b}{:}\mathtt{x}$), we drop events $e_7, \dots, e_{12}$ and obtain the middle branching pomset in Figure 8, with events $e_1, \dots, e_6, e_{13}, e_{14}$ and branching structure $\{e_1, e_2, e_{13}, e_{14}, \{\{e_3, e_4\}, \{e_5, e_6\}\}\}$.

We now formally define branching pomsets.

Definition 1 (Branching pomset)**.** A branching pomset is a four-tuple $R = \langle E, \leq, \lambda, \mathcal{B} \rangle$, where E is a set of events, $\leq \subseteq E \times E$ is such that $\leq^\star$ (the transitive closure of $\leq$) is a partial order on events, $\lambda : E \mapsto \mathcal{L}$ is a labelling function assigning an action to every event, 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 use $R.E$, $R.{\leq}$, $R.\lambda$ and $R.\mathcal{B}$ to refer to the components of R.

Semantics To fire an event in a branching pomset, on top of being minimal it must also be *active*, i.e., it must not be inside any choice. In other words: it must be a child of the branching structure's root node. We thus define a set of refinement rules in Figure 7a, written $R \sqsupseteq R'$, which can be used to resolve choices and move events upwards in the branching structure.

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 the pomset in Figure 6 we can fire the event $\mathtt{ab!x}$ in its first branch; alternatively, by applying it to the pomset in Figure 5 we can discard one branch of the choice and then fire the event $\mathtt{cd!x}$, which is now minimal. The latter use corresponds with the partial termination rules for choreographies. The fourth rule, CONGR is used for more fine-grained partial termination. To make the event $\mathtt{da!x}$ minimal in Figure 6 we could resolve two choices with CHOICE (and TRANS). However, as the rules for partial termination tell us, it is unnecessary to resolve the outer choice. Instead, we can apply CHOICE to both inner choices and apply CONGR to the outer choice to update it without unnecessarily resolving it. Finally, the fifth rule overloads the refinement notation to also apply to branching pomsets themselves: if $R.\mathcal{B}$ can refine to some $\mathcal{B}'$, then R itself can refine to a derived branching pomset with branching structure $\mathcal{B}'$, whose events are restricted to those occurring in $\mathcal{B}'$ and likewise for $\leq$ and λ.

The reduction and termination rules are defined in Figure 7b. The first rule simply states that a pomset can terminate if its branching structure can reduce to the empty set. The second rule defines the conditions for *enabling* an event e, written $R \xrightarrow{\checkmark_e} R'$. A branching pomset R can enable e by refining to R' if e is both minimal and active in R' ($e \in$ a-min(R')), and if there is no other refinement in between in which e is already minimal and active. In other words, R may only refine as far as strictly necessary to enable e. This rule implements the same idea as partial termination, with the subtle difference that, whereas partial termination tries to remove any occurrence of a participant, in this case e is actually an event in R itself. As the two notions are very similar, we use the same notation for enabling events in branching pomsets as for partial termination. Finally, the last two rules state that, if R can enable e by refining to R', then it can fire e by reducing to $R' - e$, which is the branching pomset obtained by removing e from R' (Figure 7c). This reduction is defined both on e's label and on the event itself, the latter for internal use in proofs since $\lambda(e)$ is typically not unique but e is.

Example 3.

$$\frac{}{\mathcal{B} \sqsupseteq \mathcal{B}}[\textsc{Refl}] \quad \frac{\mathcal{B} \sqsupseteq \mathcal{B}' \sqsupseteq \mathcal{B}''}{\mathcal{B} \sqsupseteq \mathcal{B}''}[\textsc{Trans}] \quad \frac{i \in \{1,2\}}{\{\{\mathcal{B}_1,\mathcal{B}_2\}\} \cup \mathcal{B} \sqsupseteq \mathcal{B}_i \cup \mathcal{B}}[\textsc{Choice}]$$

$$\frac{\mathcal{B}_1 \sqsupseteq \mathcal{B}_1' \quad \mathcal{B}_2 \sqsupseteq \mathcal{B}_2'}{\{\{\mathcal{B}_1,\mathcal{B}_2\}\} \cup \mathcal{B} \sqsupseteq \{\{\mathcal{B}_1',\mathcal{B}_2'\}\} \cup \mathcal{B}}[\textsc{Congr}] \quad \frac{R.\mathcal{B} \sqsupseteq \mathcal{B}'}{R \sqsupseteq R[\mathcal{B}']}$$

(a) Refinement rules, where we assume for CHOICE and CONGR that $\{\mathcal{B}_1,\mathcal{B}_2\} \notin \mathcal{B}$.

$$\frac{R.\mathcal{B} \sqsupseteq \emptyset}{R\downarrow} \quad \frac{R \sqsupseteq R' \quad e \in \text{a-min}(R') \quad \forall R'' : R \sqsupseteq R'' \sqsupset R' \Rightarrow e \notin \text{a-min}(R'')}{R \xrightarrow{\checkmark_e} R'} \quad \frac{R \xrightarrow{\checkmark_e} R'}{R \xrightarrow{e} R' - e} \quad \frac{R \xrightarrow{e} R'}{R \xrightarrow{\lambda(e)} R'}$$

(b) Reduction and termination rules.

$$\begin{aligned}
\langle E, \leq, \lambda, \mathcal{B}\rangle[\mathcal{B}'] &= \langle E|_{\mathcal{B}'}, \leq|_{\mathcal{B}'}, \lambda|_{\mathcal{B}'}, \mathcal{B}'\rangle \\
X|_{\mathcal{B}} &= \text{restricts } X \text{ only to the events in } \mathcal{B} \\
\text{a-min}(R) &= \{e \in R.E \mid \nexists e' \in R.E : e' < e\} \wedge e \in R.\mathcal{B} \\
\hat{e} - e &= \hat{e} \\
\{\mathcal{C}_1, \ldots, \mathcal{C}_n\} - e &= \begin{cases} \{\mathcal{C}_1, \ldots, \mathcal{C}_{i-1}, \mathcal{C}_{i+1}, \ldots, \mathcal{C}_n\} & \text{if } \mathcal{C}_i = e \\ \{\mathcal{C}_1 - e, \ldots, \mathcal{C}_n - e\} & \text{otherwise} \end{cases} \\
\{\mathcal{B}_1, \mathcal{B}_2\} - e &= \{\mathcal{B}_1 - e, \mathcal{B}_2 - e\} \\
R - e &= R[R.\mathcal{B} - e]
\end{aligned}$$

(c) Operations on branching pomsets.

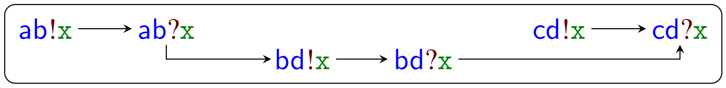
Figure 7: Semantics of branching pomsets.

- $R \xrightarrow{\checkmark_e} R'$, where R is the branching pomset in Figure 5, R' is the topmost branching pomset in Figure 8 and e is the event with label `cd!x`.
- $R \xrightarrow{\checkmark_e} R'$, where R is the branching pomset in Figure 6, R' is the middle branching pomset in Figure 8 and e is the event with label `ab!x`.
- $R \xrightarrow{\checkmark_e} R'$, where R is the branching pomset in Figure 6, R' is the middle branching pomset in Figure 8 and e is the event with label `da!x`.

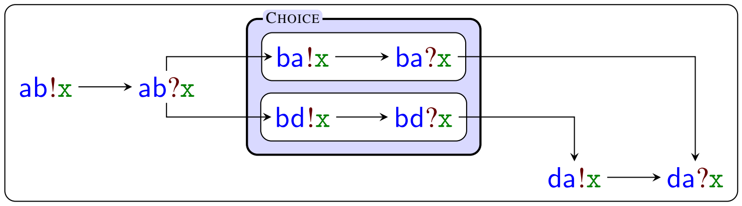
4 Branching pomsets for choreographies

In this section we formalise the construction of a branching pomset for a choreography c and we show that the pomset semantics for the branching pomset are bisimilar to the operational semantics for c.

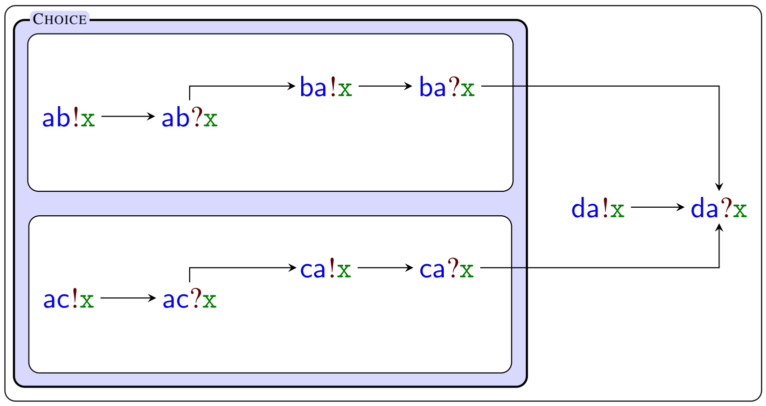
We have given examples of choreographies and corresponding branching pomsets in Figures 5 and 6. Formally, the rules for the construction of a branching pomset for a choreography c, written $[\![c]\!]$, are defined in Figure 9. Most rules are as expected. We highlight the rules for operators.



Obtained by applying CHOICE to the pomset in Figure 5.



Obtained by applying CHOICE to the outer choice of the pomset in Figure 6.



Obtained by applying CONGR to the outer choice and CHOICE to both inner choices of the pomset in Figure 6.

Figure 8: Three refined pomsets.

- The rule for parallel composition ($[\![c_1 \parallel c_2]\!]$) takes the pairwise union of all components.
- The rule for sequential composition ($[\![c_1\,;c_2]\!]$) also adds dependencies to ensure that, for every a, all events with subject a in $[\![c_1]\!]$ (denoted E_{1_a}) must precede all events with subject a in $[\![c_2]\!]$. This matches the reduction rule for weak sequential composition of choreographies (Figure 4a), as events in $[\![c_2]\!]$ are only required to wait for events in $[\![c_1]\!]$ whose subject is the same.
- The rule for choice ($[\![c_1 + c_2]\!]$) adds a single top-level choice in the branching structure to choose between the pomsets for c_1 and c_2.
- The rule for loops ($[\![c^*]\!]$) encodes a loop as a choice between terminating ($\mathbf{0}$) and unfolding one iteration of the loop ($c\,;c^*$). This results in a pomset of infinite size. We note that our theoretical results still hold even on infinite pomsets, but that any analysis of an infinite pomset 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.

As an example, we construct part of the branching pomset in Figure 5: $(\mathsf{b}\to\mathsf{c}{:}\mathtt{x} + \mathsf{b}\to\mathsf{d}{:}\mathtt{x})\,;\mathsf{c}\to\mathsf{d}{:}\mathtt{x}$ (thus omitting $\mathsf{a}\to\mathsf{b}{:}\mathtt{x}$). Let $[\![\mathsf{b}\to\mathsf{c}{:}\mathtt{x}]\!] = \langle\{e_1,e_2\},\leq_1,\lambda_1,\{e_1,e_2\}\rangle$, $[\![\mathsf{b}\to\mathsf{d}{:}\mathtt{x}]\!] = \langle\{e_3,e_4\},\leq_2,\lambda_2,\{e_3,e_4\}\rangle$ and $[\![\mathsf{c}\to\mathsf{d}{:}\mathtt{x}]\!] = \langle\{e_5,e_6\},\leq_3,\lambda_3,\{e_5,e_6\}\rangle$ as in Figure 9. First, $[\![\mathsf{b}\to\mathsf{c}{:}\mathtt{x} + \mathsf{b}\to\mathsf{d}{:}\mathtt{x}]\!] = \langle\{e_1,\dots,e_4\},\leq_1 \cup \leq_2,\lambda_1\cup\lambda_2,\{\{\{e_1,e_2\},\{e_3,e_4\}\}\}\rangle$; this is the pairwise union of the first three components, with the

$$
\begin{aligned}
[\![\mathbf{0}]\!] &= \langle \emptyset, \emptyset, \emptyset, \emptyset \rangle \\
[\![\mathsf{a}{\to}\mathsf{b}{:}\mathtt{x}]\!] &= \langle \{e_1, e_2\}, \{e_1 \le e_1, e_1 \le e_2, e_2 \le e_2\}, \{e_1 \mapsto \mathsf{ab}!\mathtt{x}, e_2 \mapsto \mathsf{ab}?\mathtt{x}\}, \{e_1, e_2\} \rangle \\
[\![\mathsf{ab}?\mathtt{x}]\!] &= \langle \{e\}, \{e \le e\}, \{e \mapsto \mathsf{ab}?\mathtt{x}\}, \{e\} \rangle \\
[\![c_1 \dagger c_2]\!] &= [\![c_1]\!] \dagger [\![c_2]\!] \text{ for } \dagger \in \{;, +, \|\} \\
[\![c^*]\!] &= [\![(c\,; c^*) + \mathbf{0}]\!]
\end{aligned}
$$

$$
\begin{aligned}
R_1\,; R_2 &= \langle E_1 \cup E_2, \le_1 \cup \le_2 \cup \textstyle\bigcup_{\mathsf{a} \in \mathcal{A}} E_{1_{\mathsf{a}}} \times E_{2_{\mathsf{a}}}, \lambda_1 \cup \lambda_2, \mathcal{B}_1 \cup \mathcal{B}_2 \rangle \\
R_1 + R_2 &= \langle E_1 \cup E_2, \le_1 \cup \le_2, \lambda_1 \cup \lambda_2, \{\{\mathcal{B}_1, \mathcal{B}_2\}\} \rangle \\
R_1 \parallel R_2 &= \langle E_1 \cup E_2, \le_1 \cup \le_2, \lambda_1 \cup \lambda_2, \mathcal{B}_1 \cup \mathcal{B}_2 \rangle
\end{aligned}
$$

Figure 9: Pomset interpretation of choreographies, where $R_i = \langle E_i, \le_i, \lambda_i, \mathcal{B}_i \rangle$ for $i \in \{1,2\}$, $\mathcal{A}$ is the set of all participants ($\mathsf{a}, \mathsf{b}, \ldots$) and $E_{i_{\mathsf{a}}}$ is the subset of events in E_i with subject a.

branching structure adding a choice between the two branches. Then $[\![(\mathsf{b}{\to}\mathsf{c}{:}\mathtt{x} + \mathsf{b}{\to}\mathsf{d}{:}\mathtt{x})\,; \mathsf{c}{\to}\mathsf{d}{:}\mathtt{x}]\!] = \langle \{e_1, \ldots, e_6\}, \le_1 \cup \le_2 \cup \le_3 \cup \{e_2 \le e_5, e_4 \le e_6\}, \lambda_1 \cup \lambda_2 \cup \lambda_2, \{e_5, e_6, \{\{e_1, e_2\}, \{e_3, e_4\}\}\} \rangle$; again, this is the pairwise union of all components, with the addition of two dependencies: $e_2 \le e_5$ represents the arrow in Figure 5 from $\mathsf{bc}?\mathtt{x}$ to $\mathsf{cd}!\mathtt{x}$ as they both have subject c, $e_4 \le e_6$ represents the arrow from $\mathsf{bd}?\mathtt{x}$ to $\mathsf{cd}?\mathtt{x}$ as they both have subject d. There are no direct dependencies between e_1 ($\mathsf{bc}!\mathtt{x}$) or e_3 ($\mathsf{bd}!\mathtt{x}$) and either e_5 or e_6, as the latter two do not have subject b.

Bisimulation For any given a choreography c we can derive two labelled transition systems: one from the operational semantics in Figure 4 over c, and one from the pomset semantics in Figure 7 over the branching pomset $[\![c]\!]$ produced by the rules in Figure 9. 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). Two systems are bisimilar if each of them can simulate the other, i.e., if they cannot be distinguished from each other just by looking at their behaviour. 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 4.

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

Formally, two transition systems A_1, A_2 are bisimilar, written $A_1 \sim A_2$, if there exists a bisimulation relation $\mathcal{R}$ relating the states of A_1 and A_2 which relates their initial states [14]. 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.

- If $p\!\downarrow$ then $q\!\downarrow$, and vice-versa.

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 $c \sim [\![c]\!]$ for all (dependently guarded) choreographies c: we define a relation $\mathcal{R} = \{\langle c, [\![c]\!]\rangle \mid c \text{ is a dependently guarded choreography}\}$ relating all dependently guarded choreographies with their interpretation as branching pomset by the rules in Figure 9. We then show that:

- If $c \xrightarrow{\ell} c'$ then $[\![c]\!] \xrightarrow{\ell} [\![c']\!]$ (Lemma 2).
- If $[\![c]\!] \xrightarrow{\ell} R'$ then $c \xrightarrow{\ell} c'$ such that $R' = [\![c']\!]$ (Lemma 3).
- If $c\!\downarrow$ then $[\![c]\!]\!\downarrow$ (Lemma 4).
- If $[\![c]\!]\!\downarrow$ then $c\!\downarrow$ (Lemma 5).

Together these lemmas prove that $c \sim [\![c]\!]$ for all dependently guarded c (Theorem 6). 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 Lemma 2 and if $[\![c_1 \,; c_2]\!] \xrightarrow{e} R'$ where e is an event in $[\![c_2]\!]$ in Lemma 3. To prove these specific cases we need to show a correspondence between partial termination and enabling events. We do this with Lemma 1, in which we show two directions simultaneously. If the choreography c_1 can partially terminate for the subject of an action ℓ in c_2 then the branching pomset $[\![c_1 \,; c_2]\!]$ can enable the corresponding event. Conversely, if $[\![c_1 \,; c_2]\!]$ can enable some event in $[\![c_2]\!]$ then the choreography c_1 can partially terminate for the subject of its label. When proving these cases in Lemmas 2 and 3, we then only have to show that the preconditions of Lemma 1 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 can be found in Appendix A, the omitted technical lemmas in Appendix B.

Lemma 1. *Let c_1 and c_2 be dependently guarded choreographies. Let $c_2 \xrightarrow{\ell} c_2'$ and $[\![c_2]\!] \xrightarrow{\checkmark_e} R_2'$ such that $\lambda(e) = \ell$ and $[\![c_2']\!] = R_2' - e$.*

(a) If $c_1 \xrightarrow{\checkmark_\ell} c_1'$ then $[\![c_1 \,; c_2]\!] \xrightarrow{\checkmark_e} [\![c_1']\!] \,; R_2'$.

(b) If $[\![c_1 \,; c_2]\!] \xrightarrow{\checkmark_e} R_1' \,; R_2'$ then $c_1 \xrightarrow{\checkmark_{\lambda(e)}} c_1'$ and $[\![c_1']\!] = R_1'$.

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 $[\![c_1']\!] \,; R_2'$ and that $[\![c_1']\!] \,; R_2'$ is the first refinement for which this is true, and then applying the second rule in Figure 7b; every case in (b) consists of showing that $[\![c_3 \,; c_2]\!] \xrightarrow{\checkmark_e} [\![c_3']\!] \,; R_2'$ 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 $c_3 \xrightarrow{\checkmark_\ell} c_3'$ and $c_4 \xrightarrow{\checkmark_\ell} c_4'$, and finally applying the partial termination rules in Figure 4c. □

Lemma 2. *Let c be a dependently guarded choreography. If $c \xrightarrow{\ell} c'$ then $[\![c]\!] \xrightarrow{\ell} [\![c']\!]$.*

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 Lemma 1 hold by the induction hypothesis and the result swiftly follows. All other cases are straightforward. □

Lemma 3. *Let c be a dependently guarded choreography. If $[\![c]\!] \xrightarrow{\ell} R'$ for some R' then $c \xrightarrow{\ell} c'$ such that $R' = [\![c']\!]$.*

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 $R' = R_1' \,;\, [\![c_1^*]\!]$ such that $[\![c_1]\!] \xrightarrow{\ell} R_1'$. It then follows from the induction hypothesis that $c_1 \xrightarrow{\ell} c_1'$ such that $[\![c_1']\!] = R_1'$. The remainder is straightforward.
- If $c = c_1 \,;\, c_2$ then $[\![c]\!] = [\![c_1]\!] \,;\, [\![c_2]\!]$. If e is an event in $[\![c_2]\!]$ then we proceed to show that $[\![c_2]\!] \xrightarrow{\ell} R_2'$, at which point we can apply the induction hypothesis. We have then satisfied the premises of Lemma 1. The remainder is straightforward.

All other cases are straightforward. □

Lemma 4. *Let c be a dependently guarded choreography. If $c\downarrow$ then $[\![c]\!]\downarrow$.*

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

Lemma 5. *Let c be a dependently guarded choreography. If $[\![c]\!]\downarrow$ then $c\downarrow$.*

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

Theorem 6. *Let c be a dependently guarded choreography. Then $c \sim [\![c]\!]$.*

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

- If $c \xrightarrow{\ell} c'$ then $R \xrightarrow{\ell} R'$ and $\langle c', R'\rangle \in \mathcal{R}$ (Lemma 2).
- If $R \xrightarrow{\ell} R'$ then $c \xrightarrow{\ell} c'$ and $\langle c', R'\rangle \in \mathcal{R}$ (Lemma 3).
- If $c\downarrow$ then $R\downarrow$ (Lemma 4).
- If $R\downarrow$ then $c\downarrow$ (Lemma 5).

Then $\mathcal{R}$ is a bisimulation relation and $c \sim [\![c]\!]$ ([14]). □

5 Conclusion

We have defined a choreography language and its operational semantics (Figures 3 and 4) using the weak sequential composition and partial termination of Rensink and Wehrheim [13], which is novel in the context of choreographies. We have defined a model, branching pomsets (Definition 1), which can compactly represent both concurrency and choices, and have defined its semantics (Figure 7). We have shown that we can use branching pomsets to model choreographies (Figure 9) and that this model is behaviourally equivalent to the operational semantics (Theorem 6).

We believe that branching pomsets can be further improved. We mention three points in particular and then discuss related work.

Binary choices Our branching structure $\mathcal{B}$ only supports 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 $[\![c_1]\!]$, $[\![c_2]\!]$ and $[\![c_3]\!]$ 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 7a), in particular CHOICE. A naive change would be to simply have this rule use $i \in \{1, \ldots, n\}$ and $\{\{\mathcal{B}_1, \ldots, \mathcal{B}_n\}\}$ 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 branching pomset can refine to $[\![c_1 + c_2]\!]$, but a branching pomset 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 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 $\{\{\mathcal{B}_1, \ldots, \mathcal{B}_m\}\}$ to refine to choice between an arbitrary (non-empty) subset of its branches.

Partial order In Definition 1, $\leq$ is defined as a relation on events such that its transitive closure is a partial order, rather than $\leq$ being a partial order itself as it is in traditional pomsets. The need for this change arises from the update rule $R[\mathcal{B}]$ (Figure 7c) in our use case as choreographies. Consider the branching pomset in Figure 5. To match the operational semantics, we should be able to refine this pomset by discarding the $\mathtt{b}{\rightarrow}\mathtt{c}{:}\mathtt{x}$ branch of the choice, after which $\mathtt{cd!x}$ should be minimal. In our current rules the events $\mathtt{bc!x}$ and $\mathtt{bc?x}$ are removed along with their entries in $\leq$ and then $\mathtt{cd!x}$ is minimal. However, if $\leq$ is a partial order, then since a partial order is transitive $\leq$ would also contain the entries $\mathtt{ab!x} \leq \mathtt{cd!x}$ and $\mathtt{ab?x} \leq \mathtt{cd!x}$ and, since these entries do not contain $\mathtt{bc!x}$ or $\mathtt{bc?x}$ but are obtained by transitivity, they are not removed. Consequently, there would be no refinement that enables $\mathtt{cd!x}$.

In general, if $R_1 \sqsupseteq R_1'$ and $R_2 \sqsupseteq R_2'$ then it would not necessarily be true that $R_1 ; R_2 \sqsupseteq R_1' ; R_2'$, as $R_1 ; R_2$ may contain dependencies obtained by transitivity which would still be present in its updated version but which cannot be derived in $R_1' ; R_2'$. We have no ready alternative. In the case of choreographies it may suffice to provide a more sophisticated update rule which properly trims these unwanted dependencies, but since this relies on knowledge of how these dependencies were derived from choreographies it is difficult to see how this could be applied to branching pomsets in general.

Loops In Figure 9 a loop c^* is encoded by infinitely unfolding it. As such, branching pomsets do not currently provide a finite representation of infinite choreographies. This remains a topic for future work, for which we envision two 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. Another possibility would be to explore the approach used in message sequence chart graphs [1] and add a graph structure on top of the branching structure.

Related work 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 [9, 1], multiparty session types [7, 8], and choreographic programs [2, 5]. 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 actions as interactions and their (parallel, sequential, and choice) composition.

Pomsets were initially introduced by Pratt [12] for concurrent models and have been widely used, e.g., in the context of message sequence charts by Katoen and Lambert [10]. Recently Guanciale and

Tuosto proposed two semantic frameworks for choreographies, one of which uses sets of pomsets [15]. 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 semantic framework that avoids exponential growth in the number of choices while still being based on pomsets. In another recent paper they use pomsets to reason over choreography realisability [6]. This demonstrates the potential of using pomsets for semantic analysis, and we are investigating how to use our framework for similar analysis.

Other related work includes the usage of event structures in the context of binary session types by Castellan and Yoshida [3] and multiparty by Castellani et al. [4]. Event structures and branching pomsets both feature a set of events with a causality relation and a choice mechanism. The main difference between the two approaches is in the choice mechanism. Event structures are based on a conflict relation on events, where two events in conflict cannot occur together in an execution and one of the two must be chosen. In contrast, we structure events in branching pomsets hierarchically. Given a branching pomset, one may construct an event structure by defining its conflict relation as all pairs of events that belong to different branches of some choice in the branching structure.

Acknowledgments

Funding by G. Cledou and J. Proença: European Regional Development Fund (ERDF), Operational Programme for Competitiveness and Internationalisation (COMPETE 2020), and Fundação para a Ciência e a Tecnologia (FCT): POCI-01-0145-FEDER-029946 (DaVinci). *Funding by S. Jongmans:* Netherlands Organisation of Scientific Research: 016.Veni.192.103. *Funding by J. Proença:* FCT, within the CISTER Research Unit: UIDB/04234/2020. FCT, I.P. (Portuguese Foundation for Science and Technology): PTDC/CCI-COM/4280/2021 (IBEX). ERDF and FCT, Portugal 2020 Partnership Agreement, Norte Portugal Regional Operational Programme (NORTE 2020): NORTE-01-0145-FEDER-028550 (REASSURE). ECSEL Joint Undertaking (JU): grant agreement No 876852 (VALU3S).

References

[1] Rajeev Alur, Kousha Etessami & Mihalis Yannakakis (2005): *Realizability and verification of MSC graphs*. *Theor. Comput. Sci.* 331(1), pp. 97–114.

[2] Marco Carbone & Fabrizio Montesi (2013): *Deadlock-freedom-by-design: multiparty asynchronous global programming*. In: *POPL*, ACM, pp. 263–274.

[3] Simon Castellan & Nobuko Yoshida (2019): *Two sides of the same coin: session types and game semantics: a synchronous side and an asynchronous side*. *Proc. ACM Program. Lang.* 3(POPL), pp. 27:1–27:29, doi:10.1145/3290340. Available at `https://doi.org/10.1145/3290340`.

[4] Ilaria Castellani, Mariangiola Dezani-Ciancaglini & Paola Giannini (2019): *Event Structure Semantics for Multiparty Sessions*. In Michele Boreale, Flavio Corradini, Michele Loreti & Rosario Pugliese, editors: *Models, Languages, and Tools for Concurrent and Distributed Programming - Essays Dedicated to Rocco De Nicola on the Occasion of His 65th Birthday*, *Lecture Notes in Computer Science* 11665, Springer, pp. 340–363, doi:10.1007/978-3-030-21485-2_19. Available at `https://doi.org/10.1007/978-3-030-21485-2_19`.

[5] Luís Cruz-Filipe & Fabrizio Montesi (2020): *A core model for choreographic programming*. *Theor. Comput. Sci.* 802, pp. 38–66.

[6] Roberto Guanciale & Emilio Tuosto (2019): *Realisability of pomsets*. *J. Log. Algebraic Methods Program.* 108, pp. 69–89, doi:10.1016/j.jlamp.2019.06.003. Available at `https://doi.org/10.1016/j.jlamp.2019.06.003`.

[7] Kohei Honda, Nobuko Yoshida & Marco Carbone (2008): *Multiparty asynchronous session types*. In: *POPL*, ACM, pp. 273–284.

[8] Kohei Honda, Nobuko Yoshida & Marco Carbone (2016): *Multiparty Asynchronous Session Types*. *J. ACM* 63(1), pp. 9:1–9:67.

[9] ITU (2011): *ITU-T recommendation Z.120. Message Sequence Chart (MSC)*.

[10] Joost-Pieter Katoen & Lennard Lambert (1998): *Pomsets for MSC*. In Hartmut König & Peter Langendörfer, editors: *Formale Beschreibungstechniken für verteilte Systeme, 8. GI/ITG-Fachgespräch, Cottbus, 4. und 5. Juni 1998*, Verlag Shaker, pp. 197–207.

[11] Rumyana Neykova & Nobuko Yoshida (2017): *Let it recover: multiparty protocol-induced recovery*. In Peng Wu & Sebastian Hack, editors: *Proceedings of the 26th International Conference on Compiler Construction, Austin, TX, USA, February 5-6, 2017*, ACM, pp. 98–108. Available at `http://dl.acm.org/citation.cfm?id=3033031`.

[12] Vaughan R. Pratt (1986): *Modeling concurrency with partial orders*. *Int. J. Parallel Program.* 15(1), pp. 33–71, doi:10.1007/BF01379149. Available at `https://doi.org/10.1007/BF01379149`.

[13] Arend Rensink & Heike Wehrheim (2001): *Process algebra with action dependencies*. *Acta Informatica* 38(3), pp. 155–234, doi:10.1007/s002360100070. Available at `https://doi.org/10.1007/s002360100070`.

[14] Davide Sangiorgi (2011): *Introduction to bisimulation and coinduction*. Cambridge University Press.

[15] Emilio Tuosto & Roberto Guanciale (2018): *Semantics of global view of choreographies*. *J. Log. Algebraic Methods Program.* 95, pp. 17–40, doi:10.1016/j.jlamp.2017.11.002. Available at `https://doi.org/10.1016/j.jlamp.2017.11.002`.

A Proofs from the paper

Lemma 1. *Let c_1 and c_2 be dependently guarded choreographies. Let $c_2 \xrightarrow{\ell} c_2'$ and $[\![c_2]\!] \xrightarrow{\checkmark_e} R_2'$ such that $\lambda(e) = \ell$ and $[\![c_2']\!] = R_2' - e$.*

(a) If $c_1 \xrightarrow{\checkmark_\ell} c_1'$ then $[\![c_1\,;c_2]\!] \xrightarrow{\checkmark_e} [\![c_1']\!]\,;R_2'$.

(b) If $[\![c_1\,;c_2]\!] \xrightarrow{\checkmark_e} R_1'\,;R_2'$ then $c_1 \xrightarrow{\checkmark_{\lambda(e)}} c_1'$ and $[\![c_1']\!] = R_1'$.

Proof. This is a proof by induction on the structure of c_1. We assume both (a) and (b) to hold for all subexpressions of c_1.

(a)

- If $c_1 = \mathbf{0}$ then $[\![c_1]\!] = [\![c_1']\!] = \langle \emptyset, \emptyset, \emptyset, \emptyset \rangle$ and $[\![c_1\,;c_2]\!] = [\![c_2]\!]$ so the result holds trivially.
- If $c_1 = \mathtt{a}{\rightarrow}\mathtt{b}{:}\mathtt{x}$ then by Fig. 4 $subj(\ell) \notin \{\mathtt{a},\mathtt{b}\}$ and $c_1' = c_1$. By Fig. 9 the construction of $[\![c_1\,;c_2]\!]$ adds no dependencies between events in $[\![c_1]\!]$ and e, so $[\![c_1\,;c_2]\!] \xrightarrow{\checkmark_e} [\![c_1]\!]\,;R_2' = [\![c_1']\!]\,;R_2'$.
- If $c_1 = \mathtt{ab}?\mathtt{x}$ then we proceed analogously to the previous case.
- If $c_1 = c_3 \dagger c_4$ for $\dagger \in \{;, \|\}$ then by Fig. 4 $c_3 \xrightarrow{\checkmark_\ell} c_3'$ and $c_4 \xrightarrow{\checkmark_\ell} c_4'$ and $c_1' = c_3' \dagger c_4'$. By the induction hypothesis (a) $[\![c_3\,;c_2]\!] \xrightarrow{\checkmark_e} [\![c_3']\!]\,;R_2'$ and $[\![c_4\,;c_2]\!] \xrightarrow{\checkmark_e} [\![c_4']\!]\,;R_2'$. By Fig. 7 $[\![c_2]\!] \sqsupseteq R_2'$, $[\![c_3]\!] \sqsupseteq [\![c_3']\!]$ and $[\![c_4]\!] \sqsupseteq [\![c_4']\!]$. By Lemma 8(i,iii) $[\![(c_3 \dagger c_4)\,;c_2]\!] \sqsupseteq ([\![c_3']\!] \dagger [\![c_4']\!])\,;R_2'$. Since $e \in \text{a-min}(R_2')$, $e \in \text{a-min}([\![c_3']\!]\,;R_2')$ and $e \in \text{a-min}([\![c_4']\!]\,;R_2')$, it follows that $e \in \text{a-min}(([\![c_3']\!] \dagger [\![c_4']\!])\,;R_2')$. Suppose there exists some R'' such that $([\![c_3]\!] \dagger [\![c_4]\!])\,;[\![c_2]\!] \sqsupseteq R'' \sqsupset ([\![c_3]\!] \dagger [\![c_4]\!])\,;R_2'$. If $e \in \text{a-min}(R'')$ then it follows from Lemma 8(iv) that either $[\![c_3]\!]\,;[\![c_2]\!] \sqsupseteq R_3' \sqsupset [\![c_3']\!]\,;R_2'$ and $e \in \text{a-min}(R_3')$ or analogously for c_4. This contradicts our observation that $[\![c_3\,;c_2]\!] \xrightarrow{\checkmark_e} [\![c_3']\!]\,;R_2'$, or analogously for c_4. We conclude that $e \notin \text{a-min}(R'')$ and then by Fig. 7 $[\![c_1\,;c_2]\!] \xrightarrow{\checkmark_e} [\![c_1']\!]\,;R_2'$.
- If $c_1 = c_3 + c_4$, we can distinguish three cases:
  - If $c_3 \xrightarrow{\checkmark_\ell} c_3'$ and $c_4 \xrightarrow{\checkmark_\ell} c_4'$ then $c_1' = c_3' + c_4'$. We then proceed analogously to the previous case, applying Lemma 8(ii,v) instead of Lemma 8(i,iii,iv).
  - If $c_3 \xrightarrow{\checkmark_\ell} c_3'$ but $c_4 \not\xrightarrow{\checkmark_\ell}$ then $c_1' = c_3'$. By the induction hypothesis (a) $[\![c_3\,;c_2]\!] \xrightarrow{\checkmark_e} [\![c_3']\!]\,;R_2'$, from which it follows that $e \in \text{a-min}([\![c_3']\!]\,;R_2')$. By the induction hypothesis (b) $[\![c_4\,;c_2]\!] \not\xrightarrow{\checkmark_e}$ since it would otherwise contradict the premise that $c_4 \not\xrightarrow{\checkmark_\ell}$. By Lemma 8(ii,iii) $[\![c_1\,;c_2]\!] \sqsupseteq [\![c_3']\!]\,;R_2'$. Suppose that there exists some R'' such that $[\![c_1\,;c_2]\!] \sqsupseteq R'' \sqsupset [\![c_3']\!]\,;R_2'$. By Lemma 8(iv) $R'' = R_1''\,;R_2''$ for some $[\![c_1]\!] \sqsupseteq R_1'' \sqsupseteq [\![c_3']\!]$ and $[\![c_2]\!] \sqsupseteq R_2'' \sqsupseteq R_2'$. If $R_2'' \neq R_2'$ then $e \notin \text{a-min}(R_2'')$ and $e \notin \text{a-min}(R'')$. By Lemma 8(v) either $R_1'' = R_4''$ for some $[\![c_4]\!] \sqsupseteq R_4'' \sqsupseteq [\![c_3']\!]$, which is clearly impossible, or $R_1'' = R_3''$ for some $[\![c_3]\!] \sqsupseteq R_3'' \sqsupseteq [\![c_3']\!]$, in which case $e \notin \text{a-min}(R'')$ since this would otherwise contradict $[\![c_3\,;c_2]\!] \xrightarrow{\checkmark_e} [\![c_3']\!]\,;R_2'$, or $R_1'' = R_3'' + R_4''$, in which case either $e \notin \text{a-min}(R'')$ or $e \in \text{a-min}(R_4''\,;R_2'')$, which contradicts $[\![c_4\,;c_2]\!] \not\xrightarrow{\checkmark_e}$. Then by Fig. 7 $[\![c_1\,;c_2]\!] \xrightarrow{\checkmark_e} [\![c_3']\!]\,;R_2' = [\![c_1']\!]\,;R_2'$.
  - If $c_3 \not\xrightarrow{\checkmark_\ell}$ and $c_4 \xrightarrow{\checkmark_\ell} c_4'$ then we proceed analogously to the previous case.
- If $c_1 = c_3^*$ for some c_3 then by Fig. 4 we can distinguish two cases:
  - If $c_3 \xrightarrow{\checkmark_\ell} c_3$ then $c_1' = c_1$. Since c_1 is dependently guarded, it follows that the subject of ℓ does not occur in c_3 or in c_1. Then by Fig. 9 there are no dependencies between any event in $[\![c_1]\!]$ and e in $[\![c_1\,;c_2]\!]$. It follows that $[\![c_1\,;c_2]\!] \sqsupseteq [\![c_1]\!]\,;R_2'$ and $e \in \text{a-min}([\![c_1]\!]\,;R_2')$. Since $[\![c_2]\!] \xrightarrow{\checkmark_e} R_2'$ there exists no R_2'' such that $[\![c_2]\!] \sqsupseteq R_2'' \sqsupset R_2'$ and $e \in \text{a-min}(R_2'')$. It then follows from Fig. 7 that $[\![c_1\,;c_2]\!] \xrightarrow{\checkmark_e} [\![c_1']\!]\,;R_2'$.
  - If $c_3 \not\xrightarrow{\checkmark_\ell} c_3$ then $c_1' = \mathbf{0}$. By Fig. 9 $[\![c_1]\!] = [\![(c_3\,;c_3^*) + \mathbf{0}]\!]$. By Lemma 8(ii) $[\![c_1]\!] \sqsupseteq [\![\mathbf{0}]\!]$ and then by Lemma 8(iii) $[\![c_1]\!]\,;[\![c_2]\!] \sqsupseteq [\![\mathbf{0}]\!]\,;R_2' = R_2'$. Since $[\![c_2]\!] \xrightarrow{\checkmark_e} R_2'$, by Fig. 7 $e \in \text{a-min}(R_2')$.

Suppose there exists some $R'_1 \neq [\![\mathbf{0}]\!]$ such that $[\![c_1\,;c_2]\!] \xrightarrow{\checkmark_e} R'_1\,;R'_2$. Then by the induction hypothesis (b) $c_1 \xrightarrow{\checkmark_{\lambda(e)}} c'_1$ and $[\![c'_1]\!] = R'_1$ for some c'_1. Since $R'_1 \neq [\![\mathbf{0}]\!]$, $c'_1 \neq \mathbf{0}$, which contradicts our earlier statement that $c_1 = \mathbf{0}$. We conclude that there exists no such R'_1 and that by Fig. 7 $[\![c_1\,;c_2]\!] \xrightarrow{\checkmark_e} [\![c'_1]\!]\,;R'_2$.

(b) By Lemma 8(iv) $[\![c_1]\!] \sqsupseteq R'_1$.

- If $c_1 = \mathbf{0}$ then $[\![c_1]\!] = [\![c'_1]\!] = R'_1 = \langle\emptyset,\emptyset,\emptyset,\emptyset\rangle$. By Fig. 4 $\mathbf{0} \xrightarrow{\checkmark_{\lambda(e)}} \mathbf{0}$ so the result holds trivially.
- If $c_1 = \mathtt{a}\to\mathtt{b}{:}\mathtt{x}$ then by Fig. 7 $R'_1 = [\![c_1]\!]$ since there is no rule to refine it and $subj(\lambda(e)) \notin \{\mathtt{a},\mathtt{b}\}$ since $e \in$ a-min$(R'_1\,;R'_2)$. Then by Fig. 4 $c_1 \xrightarrow{\checkmark_{\lambda(e)}} c_1 = c'_1$.
- If $c_1 = \mathtt{ab}?\mathtt{x}$ then we proceed analogously to the previous case.
- If $c_1 = c_3 \dagger c_4$ for $\dagger \in \{;,\|\}$ then by Lemma 8(iv) $R'_1 = R'_3 \dagger R'_4$ for some $[\![c_3]\!] \sqsupseteq R'_3$ and $[\![c_4]\!] \sqsupseteq R'_4$. Suppose there exists some R''_3 such that $[\![c_3\,;c_2]\!] \sqsupseteq R''_3\,;R'_2 \sqsupseteq R'_3\,;R'_2$ and $e \in$ a-min$(R''_3\,;R'_2)$. It would follow from Lemma 8(i,iii) that $[\![c_1\,;c_2]\!] \sqsupseteq (R''_3 \dagger R'_4)\,;R'_2 \sqsupset R'_1\,;R'_2$ and $e \in$ a-min$((R''_3 \dagger R'_4)\,;R'_2)$, which contradicts our premise that $[\![c_1\,;c_2]\!] \xrightarrow{\checkmark_e} R'_1\,;R'_2$. It thus follows from Fig. 7 that $[\![c_3\,;c_2]\!] \xrightarrow{\checkmark_e} R'_3\,;R'_2$ and similarly for c_4. By the induction hypothesis (b) $c_3 \xrightarrow{\checkmark_{\lambda(e)}} c'_3$ such that $[\![c'_3]\!] = R'_3$ and similarly for c_4. Then by Fig. 4 $c_3 \dagger c_4 \xrightarrow{\checkmark_{\lambda(e)}} c'_3 \dagger c'_4$.
- If $c_1 = c_3 + c_4$ then by Lemma 8(v) we can distinguish three cases:
  - If $R'_1 = R'_3 + R'_4$ for some $R'_3 \sqsupseteq [\![c_3]\!]$ and $R'_4 \sqsupseteq [\![c_4]\!]$ then by Lemma 8(iii) $[\![c_3\,;c_2]\!] \sqsupseteq R'_3\,;R'_2$ and similarly for c_4. Analogously to the previous case it follows that $[\![c_3\,;c_2]\!] \xrightarrow{\checkmark_e} R'_3\,;R'_2$ and by the induction hypothesis (b) that $c_3 \xrightarrow{\checkmark_{\lambda(e)}} c'_3$ and $[\![c'_3]\!] = R'_3$, and similarly for c_4. Then by Fig. 4 $c_3 + c_4 \xrightarrow{\checkmark_{\lambda(e)}} c'_3 + c'_4$ and by Fig. 9 $[\![c'_3 + c'_4]\!] = R'_3 + R'_4$.
  - If $R'_1 \sqsupseteq [\![c_3]\!]$ then analogously to the previous case it follows that $[\![c_3\,;c_2]\!] \xrightarrow{\checkmark_e} R'_1\,;R'_2$. By the induction hypothesis (b) $c_3 \xrightarrow{\checkmark_{\lambda(e)}} c'_3$ and $R'_1 = [\![c'_3]\!]$.
    Suppose that $c_4 \xrightarrow{\checkmark_{\lambda(e)}} c'_4$ for some c'_4. Then by the induction hypothesis (a) also $[\![c_4\,;c_2]\!] \xrightarrow{\checkmark_e} [\![c'_4]\!]\,;R'_2$. It would follow from Fig. 7 that $[\![(c_3 + c_4)\,;c_2]\!] \xrightarrow{\checkmark_e} ([\![c'_3]\!] + [\![c'_4]\!])\,;R'_2$. However, since $([\![c'_3]\!] + [\![c'_4]\!])\,;R'_2 \sqsupseteq [\![c'_3]\!]\,;R'_2$ this contradicts our premise that $R'_1 = [\![c'_3]\!]$. We conclude that $c_4 \not\xrightarrow{\checkmark_{\lambda(e)}}$ and then by Fig. 4 $c_3 + c_4 \xrightarrow{\checkmark_{\lambda(e)}} c'_3$.
  - If $R'_1 \sqsupseteq [\![c_4]\!]$ then we proceed analogously to the previous case.
- If $c_1 = c_3^*$ then recall that by Fig. 9 $[\![c_3^*]\!] = [\![(c_3\,;c_3^*) + \mathbf{0}]\!]$. We can distinguish two cases:
  - If $R'_1 = [\![c_1]\!]$ then analogously to the previous cases it follows that $[\![c_3\,;c_2]\!] \xrightarrow{\checkmark_e} [\![c_3]\!]\,;R'_2$ and by the induction hypothesis (b) $c_3 \xrightarrow{\checkmark_{\lambda(e)}} c_3$. Then by Fig. 4 $c_1 \xrightarrow{\checkmark_{\lambda(e)}} c_1$.
  - If $R'_1 \neq [\![c_1]\!]$ then suppose that $[\![c_3\,;c_2]\!] \xrightarrow{\checkmark_e} R'_3\,;R'_2$ for some $[\![c_3]\!] \sqsupset R'_3$. It would follow from the induction hypothesis (b) that $c_3 \xrightarrow{\checkmark_{\lambda(e)}} c'_3$ such that $[\![c'_3]\!] = R'_3$. Then $c'_3 \neq c_3$, which is contradictory since $c_1 = c_3^*$ is dependently guarded. It thus follows that $[\![c_3\,;c_2]\!] \not\xrightarrow{\checkmark_e}$ and that $R'_1 = [\![\mathbf{0}]\!]$. By the induction hypothesis (a) $c_3 \not\xrightarrow{\checkmark_{\lambda(e)}}$ and then by Fig. 4 $c_1 \xrightarrow{\checkmark_{\lambda(e)}} \mathbf{0}$. □

Lemma 2. *Let c be a dependently guarded choreography. If $c \xrightarrow{\ell} c'$ then $[\![c]\!] \xrightarrow{\ell} [\![c']\!]$.*

Proof. This is a proof by structural induction on c.

- Suppose $c \in \{\mathbf{0}, \mathtt{a}\to\mathtt{b}{:}\mathtt{x}, \mathtt{ab}?\mathtt{x}\}$. Then the result holds trivially.
- Suppose $c = c_1 \parallel c_2$. If $c \xrightarrow{\ell} c'$, then without loss of generality $c_1 \xrightarrow{\ell} c'_1$ and $c' = c'_1 \parallel c_2$ (the other case is analogous). By the induction hypothesis $[\![c_1]\!] \xrightarrow{\ell} [\![c'_1]\!]$, so by Fig. 7 $[\![c_1]\!] \xrightarrow{\checkmark_e} R'$ such that $[\![c'_1]\!] = R' - e$ and $\lambda(e) = \ell$. It follows from Lemma 9(i) that $[\![c'_1]\!] \parallel [\![c_2]\!] \xrightarrow{\checkmark_e} R' \parallel [\![c_2]\!]$ and then by Fig. 7 $[\![c]\!] = [\![c_1]\!] \parallel [\![c_2]\!] \xrightarrow{\ell} (R' - e) \parallel [\![c_2]\!] = [\![c'_1]\!] \parallel [\![c_2]\!] = [\![c']\!]$.

- Suppose $c = c_1 + c_2$. If $c \xrightarrow{\ell} c'$, then without loss of generality $c_1 \xrightarrow{\ell} c'$ (the other case if analogous). By the induction hypothesis $[\![c_1]\!] \xrightarrow{\ell} [\![c']\!]$, so by Fig. 7 $[\![c_1]\!] \xrightarrow{\checkmark_e} R'$ such that $[\![c']\!] = R' - e$ and $\lambda(e) = \ell$. It follows from Lemma 9(ii) that $[\![c_1]\!] + [\![c_2]\!] \xrightarrow{\checkmark_e} R'$ and then by Fig. 7 $[\![c]\!] = [\![c_1]\!] + [\![c_2]\!] \xrightarrow{\ell} R' - e = [\![c']\!]$.
- Suppose $c = c_1^*$. If $c \xrightarrow{\ell} c'$, then $c_1 \xrightarrow{\ell} c_1'$ and $c' = c_1' \,;c_1^*$. By the induction hypothesis $[\![c_1]\!] \xrightarrow{\ell} [\![c_1']\!]$, so by Fig. 7 $[\![c_1]\!] \xrightarrow{\checkmark_e} R'$ such that $[\![c_1']\!] = R' - e$ and $\lambda(e) = \ell$. Since $[\![c_1^*]\!] = [\![(c_1\,; c_1^*) + \mathbf{0}]\!]$, it follows from Lemma 9(ii–iii) that $[\![(c_1\,; c_1^*) + \mathbf{0}]\!] \xrightarrow{\checkmark_e} R'\,; [\![c_1^*]\!]$ and then by Fig. 7 $[\![c]\!] = [\![(c_1\,; c_1^*) + \mathbf{0}]\!] \xrightarrow{\ell} (R' - e)\,; [\![c_1^*]\!] = [\![c_1']\!]\,; [\![c_1^*]\!] = [\![c']\!]$.
- Finally, suppose $c = c_1\,; c_2$. If $c \xrightarrow{\ell} c'$, we can distinguish two cases:
  - Suppose $c_1 \xrightarrow{\ell} c_1'$ and $c' = c_1'; c_2$. By the induction hypothesis $[\![c_1]\!] \xrightarrow{\ell} [\![c_1']\!]$, so by Fig. 7 $[\![c_1]\!] \xrightarrow{\checkmark_e} R'$ such that $[\![c_1']\!] = R' - e$ and $\lambda(e) = \ell$. It follows from Lemma 9(iii) that $[\![c_1]\!]\,; [\![c_2]\!] \xrightarrow{\checkmark_e} R'; [\![c_2]\!]$ and then by Fig. 7 $[\![c]\!] = [\![c_1]\!]\,; [\![c_2]\!] \xrightarrow{\ell} (R' - e)\,; [\![c_2]\!] = [\![c_1']\!]\,; [\![c_2]\!] = [\![c']\!]$.
  - Suppose $c_1 \xrightarrow{\checkmark_\ell} c_1'$, $c_2 \xrightarrow{\ell} c_2'$ and $c' = c_1'\,; c_2'$. By the induction hypothesis $[\![c_2]\!] \xrightarrow{\ell} [\![c_2']\!]$ and then it follows from Lemma 1(a) that $[\![c]\!] = [\![c_1]\!]\,; [\![c_2]\!] \xrightarrow{\checkmark_e} [\![c_1']\!]\,; R_2' \xrightarrow{\ell} [\![c_1']\!]\,; [\![c_2']\!] = [\![c']\!]$. □

Lemma 3. *Let c be a dependently guarded choreography. If $[\![c]\!] \xrightarrow{\ell} R'$ for some R' then $c \xrightarrow{\ell} c'$ such that $R' = [\![c']\!]$.*

Proof. This is a proof by structural induction on c. Let $R = [\![c]\!]$.

- Suppose $c \in \{\mathbf{0}, \mathtt{a}{\rightarrow}\mathtt{b}{:}\mathtt{x}, \mathtt{ab}?\mathtt{x}\}$. Then the result holds trivially.
- Suppose $c = c_1 \parallel c_2$. If $R \xrightarrow{\ell} R'$, then without loss of generality $[\![c_1]\!] \xrightarrow{\ell} R_1'$ and $R' = R_1' \parallel [\![c_2]\!]$ (the other case is analogous). By the induction hypothesis there exists some c_1' such that $c_1 \xrightarrow{\ell} c_1'$ such that $R_1' = [\![c_1']\!]$. Then by Fig. 4 $c_1 \parallel c_2 \xrightarrow{\ell} c_1' \parallel c_2 = c'$, and $[\![c']\!] = R'$.
- Suppose $c = c_1 + c_2$. If $R \xrightarrow{\ell} R'$, then without loss of generality $[\![c_1]\!] \xrightarrow{\ell} R'$ (the other case is analogous). By the induction hypothesis there then exists some c' such that $c_1 \xrightarrow{\ell} c'$ and $[\![c']\!] = R'$ and then by Fig. 4 $c_1 + c_2 \xrightarrow{\ell} c'$.
- Suppose $c = c_1^*$. If $R \xrightarrow{\ell} R'$, then it follows from Lemma 10 that $[\![c_1]\!] \xrightarrow{\ell} R_1'$ and $R' = R_1'\,; [\![c_1^*]\!]$. By the induction hypothesis there exists some c_1' such that $c_1 \xrightarrow{\ell} c_1'$ and $R_1' = [\![c_1']\!]$. Then by Fig. 4 $c_1^* \xrightarrow{\ell} c_1'\,; c_1^* = c'$, and $R' = [\![c']\!]$.
- Finally, suppose $c = c_1\,; c_2$. If $R \xrightarrow{\ell} R'$, then by Fig. 7 $R \xrightarrow{\checkmark_e} R''$ such that $R' = R'' - e$ and $\lambda(e) = \ell$. By Lemma 8(iv) $R'' = R_1'\,; R_2'$ for some $[\![c_1]\!] \sqsupseteq R_1'$ and $[\![c_2]\!] \sqsupseteq R_2'$. We can distinguish two cases:
  - Suppose e is an event in $[\![c_1]\!]$. Suppose $[\![c_1]\!] \sqsupseteq R_1'' \sqsupset R_1'$ for some R_1''. Then $e \notin$ a-min(R_1''). If it were, then also $[\![c_1]\!]\,; [\![c_2]\!] \sqsupseteq R_1''\,; R_2' \sqsupset R_1'\,; R_2'$ and $e \in$ a-min$(R_1''\,; R_2')$, which contradicts Fig. 7. It follows from Fig. 7 that $[\![c_1]\!] \xrightarrow{\checkmark_e} R_1'$ and $[\![c_1]\!] \xrightarrow{\ell} R_1' - e$. By the induction hypothesis there exists some c_1' such that $c_1 \xrightarrow{\ell} c_1'$ and $[\![c_1']\!] = R_1' - e$. By Fig. 4 $c_1\,; c_2 \xrightarrow{\ell} c_1'\,; c_2 = c'$ and then $R' = [\![c']\!]$.
  - Suppose e is an event in $[\![c_2]\!]$. Suppose $[\![c_2]\!] \sqsupseteq R_2'' \sqsupset R_2'$ for some R_2''. Then $e \notin$ a-min(R_2''). If it were, then also $[\![c_1]\!]\,; [\![c_2]\!] \sqsupseteq R_1'\,; R_2'' \sqsupset R_1'\,; R_2'$ and $e \in$ a-min$(R_1'\,; R_2'')$, which contradicts Fig. 7. It follows from Fig. 7 that $[\![c_2]\!] \xrightarrow{\checkmark_e} R_2'$ and $[\![c_2]\!] \xrightarrow{\ell} R_2' - e$. By the induction hypothesis there exists some c_2' such that $c_2 \xrightarrow{\ell} c_2'$ and $[\![c_2']\!] = R_2' - e$. It then follows from Lemma 1(b) that $c_1 \xrightarrow{\checkmark_\ell} c_1'$ and $[\![c_1']\!] = R_1'$. Then by Fig. 4, $c_1\,; c_2 \xrightarrow{\ell} c_1'\,; c_2' = c'$ and $[\![c']\!] = R'$. □

Lemma 4. *Let c be a dependently guarded choreography. If $c{\downarrow}$ then $[\![c]\!]{\downarrow}$.*

Proof. This is a proof by structural induction on c.

- If $c = \mathbf{0}$ then both c and $[\![c]\!]$ can terminate.
- If $c = \mathtt{a}{\to}\mathtt{b}{:}\mathtt{x}$ or $c = \mathtt{ab?x}$ then neither c or $[\![c]\!]$ can terminate.
- If $c = c_1 \dagger c_2$ for $\dagger \in \{;, \|\}$ then by Fig. 4 $c_1{\downarrow}$ and $c_2{\downarrow}$. By the induction hypothesis $[\![c_1]\!]{\downarrow}$ and $[\![c_2]\!]{\downarrow}$. By Fig. 7 $[\![c_1]\!].\mathcal{B} \sqsupseteq \emptyset$ and $[\![c_2]\!].\mathcal{B} \sqsupseteq \emptyset$. By Fig. 9 $[\![c_1 \dagger c_2]\!].\mathcal{B} = [\![c_1]\!].\mathcal{B} \cup [\![c_2]\!].\mathcal{B}$ and by Fig. 7 $[\![c_1 \dagger c_2]\!].\mathcal{B} \sqsupseteq \emptyset$ and $[\![c_1 \dagger c_2]\!]{\downarrow}$.
- If $c = c_1 + c_2$ then by Fig. 4 either $c_1{\downarrow}$ or $c_2{\downarrow}$. Without loss of generality we assume $c_1{\downarrow}$; the other case is analogous. By the induction hypothesis $[\![c_1]\!]{\downarrow}$ and by Fig. 7 $[\![c_1]\!].\mathcal{B} \sqsupseteq \emptyset$. By Fig. 7 $[\![c_1 + c_2]\!].\mathcal{B} \sqsupseteq \emptyset$ and then $[\![c_1 + c_2]\!]{\downarrow}$.
- If $c = c_1^*$ then $c{\downarrow}$ by Fig. 4. By Fig. 9 $[\![c]\!] = [\![(c_1\,; c_1^*) + \mathbf{0}]\!]$. Since $[\![\mathbf{0}]\!].\mathcal{B} = \emptyset$, it follows from Fig. 7 that $[\![c]\!].\mathcal{B} \sqsupseteq \emptyset$ and then $[\![c]\!]{\downarrow}$. □

Lemma 5. *Let c be a dependently guarded choreography. If $[\![c]\!]{\downarrow}$ then $c{\downarrow}$.*

Proof. This is a proof by structural induction on c.

- If $c = \mathbf{0}$ then both c and $[\![c]\!]$ can terminate.
- If $c = \mathtt{a}{\to}\mathtt{b}{:}\mathtt{x}$ or $c = \mathtt{ab?x}$ then neither c or $[\![c]\!]$ can terminate.
- If $c = c_1 \dagger c_2$ for $\dagger \in \{;, \|\}$ and $[\![c]\!]{\downarrow}$ then by Fig. 7 $[\![c_1 \dagger c_2]\!].\mathcal{B} \sqsupseteq \emptyset$. It follows from Lemma 7(ii) that $\emptyset = \mathcal{B}'_1 \cup \mathcal{B}'_2$ such that $[\![c_1]\!].\mathcal{B} \sqsupseteq \mathcal{B}'_1$ and $[\![c_2]\!].\mathcal{B} \sqsupseteq \mathcal{B}'_2$. It follows that $[\![c_1]\!].\mathcal{B} \sqsupseteq \emptyset$ and $[\![c_2]\!].\mathcal{B} \sqsupseteq \emptyset$, so by Fig. 7 $[\![c_1]\!]{\downarrow}$ and $[\![c_2]\!]{\downarrow}$. By the induction hypothesis $c_1{\downarrow}$ and $c_2{\downarrow}$ and then by Fig. 4 $c_1 \dagger c_2{\downarrow}$.
- If $c = c_1 + c_2$ and $c{\downarrow}$ then by Fig. 7 $[\![c_1 + c_2]\!].\mathcal{B} \sqsupseteq \emptyset$. By Fig. 9 $[\![c_1 + c_2]\!].\mathcal{B} = \{\{[\![c_1]\!].\mathcal{B}, [\![c_2]\!].\mathcal{B}\}\}$. By Lemma 7(iii) either:
  - $\emptyset = \{\{\mathcal{B}'_1, \mathcal{B}'_2\}\}$ for some $[\![c_1]\!].\mathcal{B} \sqsupseteq \mathcal{B}'_1$ and $[\![c_2]\!].\mathcal{B} \sqsupseteq \mathcal{B}'_2$, which is a clear contradiction; or
  - $[\![c_1]\!].\mathcal{B} \sqsupseteq \emptyset$, in which case $[\![c_1]\!]{\downarrow}$ and by the induction hypothesis $c_1{\downarrow}$ and then by Fig. 4 $c_1 + c_2{\downarrow}$; or
  - $[\![c_2]\!].\mathcal{B} \sqsupseteq \emptyset$, which is analogous to the previous case.
- If $c = c_1^*$ then, as in Lemma 4, both $[\![c]\!]{\downarrow}$ and $c{\downarrow}$. □

B Additional proofs

Lemma 7. *Let $\mathcal{B}_1, \mathcal{B}_2$ be branching structures.*

(i) If $\mathcal{B}_1 \sqsupseteq \mathcal{B}_1^\ddagger$ and $\mathcal{B}_1 \uplus \mathcal{B}_2$ is defined, then $\mathcal{B}_1 \cup \mathcal{B}_2 \sqsupseteq \mathcal{B}_1^\ddagger \cup \mathcal{B}_2$.

(ii) If $\mathcal{B}_1 \cup \mathcal{B}_2 \sqsupseteq \mathcal{B}^\ddagger$, then $\mathcal{B}_1 \sqsupseteq \mathcal{B}_1^\ddagger$ and $\mathcal{B}_2 \sqsupseteq \mathcal{B}_2^\ddagger$ and $\mathcal{B}_1^\ddagger \cup \mathcal{B}_2^\ddagger = \mathcal{B}^\ddagger$, for some $\mathcal{B}_1^\ddagger, \mathcal{B}_2^\ddagger$.

(iii) If $\{\{\mathcal{B}_1, \mathcal{B}_2\}\} \sqsupseteq \mathcal{B}^\ddagger$, then either $\mathcal{B}_1 \sqsupseteq \mathcal{B}_1^\ddagger$ and $\mathcal{B}_2 \sqsupseteq \mathcal{B}_2^\ddagger$ and $\{\{\mathcal{B}_1^\ddagger, \mathcal{B}_2^\ddagger\}\} = \mathcal{B}^\ddagger$, for some $\mathcal{B}_1^\ddagger, \mathcal{B}_2^\ddagger$, or $\mathcal{B}_1 \sqsupseteq \mathcal{B}^\ddagger$, or $\mathcal{B}_2 \sqsupseteq \mathcal{B}^\ddagger$.

Proof.

(i) Recall $\mathcal{B}_1 \sqsupseteq \mathcal{B}_1^\ddagger$. Then, by the definition of refinement:

  - **Base:** Refl, such that $\mathcal{B}_1 = \mathcal{B}_1^\ddagger$.
    Recall $\mathcal{B}_1 \uplus \mathcal{B}_2$ is defined. Then, by Refl, $\mathcal{B}_1 \cup \mathcal{B}_2 \sqsupseteq \mathcal{B}_1 \cup \mathcal{B}_2$. Then, $\boxed{\mathcal{B}_1 \cup \mathcal{B}_2 \sqsupseteq \mathcal{B}_1^\ddagger \cup \mathcal{B}_2}$.

- **Base:** CHOICE, such that $\mathcal{B}_1 = \{\{\hat{\mathcal{B}}_1, \dots, \hat{\mathcal{B}}_m\}\} \uplus \hat{\mathcal{B}}$ and $\mathcal{B}_1^\ddagger = \hat{\mathcal{B}}_i \cup \hat{\mathcal{B}}$ and $1 \leq i \leq m$, for some $\hat{\mathcal{B}}, \hat{\mathcal{B}}_1, \dots, \hat{\mathcal{B}}_m, i, m$.
  - Recall $\mathcal{B}_1 \uplus \mathcal{B}_2$ is defined. Then, $(\{\{\hat{\mathcal{B}}_1, \dots, \hat{\mathcal{B}}_m\}\} \uplus \hat{\mathcal{B}}) \uplus \mathcal{B}_2$ is defined. Then, $\{\{\hat{\mathcal{B}}_1, \dots, \hat{\mathcal{B}}_m\}\} \uplus (\hat{\mathcal{B}} \uplus \mathcal{B}_2)$ is defined.
  - Recall $\{\{\hat{\mathcal{B}}_1, \dots, \hat{\mathcal{B}}_m\}\} \uplus (\hat{\mathcal{B}} \uplus \mathcal{B}_2)$ is defined, and $1 \leq i \leq m$. Then, by CHOICE, $\{\{\hat{\mathcal{B}}_1, \dots, \hat{\mathcal{B}}_m\}\} \uplus (\hat{\mathcal{B}} \uplus \mathcal{B}_2) \sqsupseteq \hat{\mathcal{B}}_i \cup (\hat{\mathcal{B}} \uplus \mathcal{B}_2)$. Then, $\{\{\hat{\mathcal{B}}_1, \dots, \hat{\mathcal{B}}_m\}\} \cup (\hat{\mathcal{B}} \cup \mathcal{B}_2) \sqsupseteq \hat{\mathcal{B}}_i \cup (\hat{\mathcal{B}} \cup \mathcal{B}_2)$. Then, $(\{\{\hat{\mathcal{B}}_1, \dots, \hat{\mathcal{B}}_m\}\} \cup \hat{\mathcal{B}}) \cup \mathcal{B}_2 \sqsupseteq (\hat{\mathcal{B}}_i \cup \hat{\mathcal{B}}) \cup \mathcal{B}_2$. Then, $\boxed{\mathcal{B}_1 \cup \mathcal{B}_2 \sqsupseteq \mathcal{B}_1^\ddagger \cup \mathcal{B}_2}$.
- **Step:** TRANS, such that $\mathcal{B}_1 \sqsupseteq \mathcal{B}_1^\dagger \sqsupseteq \mathcal{B}_1^\ddagger$, for some $\mathcal{B}_1^\dagger$.
  - Recall $\mathcal{B}_1 \sqsupseteq \mathcal{B}_1^\dagger$ and $\mathcal{B}_1 \uplus \mathcal{B}_2$ is defined. Then, by induction, $\mathcal{B}_1 \cup \mathcal{B}_2 \sqsupseteq \mathcal{B}_1^\dagger \cup \mathcal{B}_2$.
  - Recall $\mathcal{B}_1 \cup \mathcal{B}_2 \sqsupseteq \mathcal{B}_1^\dagger \cup \mathcal{B}_2$. Then, $\mathcal{B}_1^\dagger \cup \mathcal{B}_2$ is defined.
  - Recall $\mathcal{B}_1^\dagger \sqsupseteq \mathcal{B}_1^\ddagger$ and $\mathcal{B}_1^\dagger \cup \mathcal{B}_2$ is defined. Then, by induction, $\mathcal{B}_1^\dagger \cup \mathcal{B}_2 \sqsupseteq \mathcal{B}_1^\ddagger \cup \mathcal{B}_2$.
  - Recall $\mathcal{B}_1 \cup \mathcal{B}_2 \sqsupseteq \mathcal{B}_1^\dagger \cup \mathcal{B}_2 \sqsupseteq \mathcal{B}_1^\ddagger \cup \mathcal{B}_2$. Then, by TRANS, $\boxed{\mathcal{B}_1 \cup \mathcal{B}_2 \sqsupseteq \mathcal{B}_1^\ddagger \cup \mathcal{B}_2}$.
- **Step:** CONGR. Similar to case CHOICE. □

(ii) Recall $\mathcal{B}_1 \cup \mathcal{B}_2 \sqsupseteq \mathcal{B}^\ddagger$. Then, by the definition of refinement:

- **Base:** REFL, such that $\mathcal{B}_1 \cup \mathcal{B}_2 = \mathcal{B}^\ddagger$.
  - By REFL, $\mathcal{B}_1 \sqsupseteq \mathcal{B}_1$. Then, $\mathcal{B}_1^\ddagger = \mathcal{B}_1$ and $\boxed{\mathcal{B}_1 \sqsupseteq \mathcal{B}_1^\ddagger}$, $\boxed{\text{for some } \mathcal{B}_1^\ddagger}$.
  - By REFL, $\mathcal{B}_2 \sqsupseteq \mathcal{B}_2$. Then, $\mathcal{B}_2^\ddagger = \mathcal{B}_2$ and $\boxed{\mathcal{B}_2 \sqsupseteq \mathcal{B}_2^\ddagger}$, $\boxed{\text{for some } \mathcal{B}_2^\ddagger}$.
  - Recall $\mathcal{B}_1 \cup \mathcal{B}_2 = \mathcal{B}^\ddagger$. Then, $\boxed{\mathcal{B}_1^\ddagger \cup \mathcal{B}_2^\ddagger = \mathcal{B}^\ddagger}$.
- **Base:** CHOICE, such that $\mathcal{B}_1 \cup \mathcal{B}_2 = \{\{\hat{\mathcal{B}}_1, \dots, \hat{\mathcal{B}}_m\}\} \uplus \hat{\mathcal{B}}$ and $\mathcal{B}^\ddagger = \hat{\mathcal{B}}_i \cup \hat{\mathcal{B}}$ and $1 \leq i \leq m$, for some $\hat{\mathcal{B}}, \hat{\mathcal{B}}_1, \dots, \hat{\mathcal{B}}_m, i, m$.
  Recall $\mathcal{B}_1 \cup \mathcal{B}_2 = \{\{\hat{\mathcal{B}}_1, \dots, \hat{\mathcal{B}}_m\}\} \uplus \hat{\mathcal{B}}$. Then:
  - **Case 1:** $\mathcal{B}_1 = \{\{\hat{\mathcal{B}}_1, \dots, \hat{\mathcal{B}}_m\}\} \uplus \hat{\mathcal{B}}'$ and $\mathcal{B}_2 = \hat{\mathcal{B}}''$ and $\hat{\mathcal{B}} = \hat{\mathcal{B}}' \cup \hat{\mathcal{B}}''$, for some $\hat{\mathcal{B}}', \hat{\mathcal{B}}''$.
    * Recall $1 \leq i \leq m$. Then, by CHOICE, $\{\{\hat{\mathcal{B}}_1, \dots, \hat{\mathcal{B}}_m\}\} \uplus \hat{\mathcal{B}}' \sqsupseteq \hat{\mathcal{B}}_i \cup \hat{\mathcal{B}}'$. Then, $\mathcal{B}_1 \sqsupseteq \hat{\mathcal{B}}_i \cup \hat{\mathcal{B}}'$. Then, $\mathcal{B}_1^\ddagger = \hat{\mathcal{B}}_i \cup \hat{\mathcal{B}}'$ and $\boxed{\mathcal{B}_1 \sqsupseteq \mathcal{B}_1^\ddagger}$, $\boxed{\text{for some } \mathcal{B}_1^\ddagger}$.
    * By REFL, $\hat{\mathcal{B}}'' \sqsupseteq \hat{\mathcal{B}}''$. Then, $\mathcal{B}_2 \sqsupseteq \hat{\mathcal{B}}''$. Then, $\mathcal{B}_2^\ddagger = \hat{\mathcal{B}}''$ and $\boxed{\mathcal{B}_2 \sqsupseteq \mathcal{B}_2^\ddagger}$, $\boxed{\text{for some } \mathcal{B}_2^\ddagger}$.
    * Recall $\mathcal{B}^\ddagger = \hat{\mathcal{B}}_i \cup \hat{\mathcal{B}}$. Then, $\mathcal{B}^\ddagger = \hat{\mathcal{B}}_i \cup \hat{\mathcal{B}}' \cup \hat{\mathcal{B}}''$. Then, $\boxed{\mathcal{B}_1^\ddagger \cup \mathcal{B}_2^\ddagger = \mathcal{B}^\ddagger}$.
  - **Case 2:** $\mathcal{B}_1 = \hat{\mathcal{B}}'$ and $\mathcal{B}_2 = \{\{\hat{\mathcal{B}}_1, \dots, \hat{\mathcal{B}}_m\}\} \uplus \hat{\mathcal{B}}''$ and $\hat{\mathcal{B}} = \hat{\mathcal{B}}' \cup \hat{\mathcal{B}}''$, for some $\hat{\mathcal{B}}', \hat{\mathcal{B}}''$. Similar to case 1.
- **Step:** TRANS, such that $\mathcal{B}_1 \cup \mathcal{B}_2 \sqsupseteq \mathcal{B}^\dagger \sqsupseteq \mathcal{B}^\ddagger$, for some $\mathcal{B}^\dagger$.
  - Recall $\mathcal{B}_1 \cup \mathcal{B}_2 \sqsupseteq \mathcal{B}^\dagger$. Then, by induction, $\mathcal{B}_1 \sqsupseteq \mathcal{B}_1^\dagger$ and $\mathcal{B}_2 \sqsupseteq \mathcal{B}_2^\dagger$ and $\mathcal{B}^\dagger = \mathcal{B}_1^\dagger \cup \mathcal{B}_2^\dagger$, for some $\mathcal{B}_1^\dagger, \mathcal{B}_2^\dagger$.
  - Recall $\mathcal{B}^\dagger \sqsupseteq \mathcal{B}^\ddagger$. Then, $\mathcal{B}_1^\dagger \cup \mathcal{B}_2^\dagger \sqsupseteq N^\ddagger$. Then, by induction, $\mathcal{B}_1^\dagger \sqsupseteq \mathcal{B}_1^\ddagger$ and $\mathcal{B}_2^\dagger \sqsupseteq \mathcal{B}_2^\ddagger$ and $\boxed{\mathcal{B}^\ddagger = \mathcal{B}_1^\ddagger \cup \mathcal{B}_2^\ddagger}$, $\boxed{\text{for some } \mathcal{B}_1^\ddagger, \mathcal{B}_2^\ddagger}$.
  - Recall $\mathcal{B}_1 \sqsupseteq \mathcal{B}_1^\dagger \sqsupseteq \mathcal{B}_1^\ddagger$. Then, by TRANS, $\boxed{\mathcal{B}_1 \sqsupseteq \mathcal{B}_1^\ddagger}$.
  - Recall $\mathcal{B}_2 \sqsupseteq \mathcal{B}_2^\dagger \sqsupseteq \mathcal{B}_2^\ddagger$. Then, by TRANS, $\boxed{\mathcal{B}_2 \sqsupseteq \mathcal{B}_2^\ddagger}$. □
- **Step:** CONGR. Similar to case CHOICE.

(iii) Recall $\{\{\mathcal{B}_1, \mathcal{B}_2\}\} \sqsupseteq \mathcal{B}^\ddagger$. Then, by the definition of refinement:

- **Base:** REFL, such that $\{\{\mathcal{B}_1, \mathcal{B}_2\}\} = \mathcal{B}^\ddagger$.
  - By REFL, $\mathcal{B}_1 \sqsupseteq \mathcal{B}_1$. Then, $\mathcal{B}_1^\ddagger = \mathcal{B}_1$ and $\boxed{\mathcal{B}_1 \sqsupseteq \mathcal{B}_1^\ddagger}$, $\boxed{\text{for some } \mathcal{B}_1^\ddagger}$.
  - By REFL, $\mathcal{B}_2 \sqsupseteq \mathcal{B}_2$. Then, $\mathcal{B}_2^\ddagger = \mathcal{B}_2$ and $\boxed{\mathcal{B}_2 \sqsupseteq \mathcal{B}_2^\ddagger}$, $\boxed{\text{for some } \mathcal{B}_2^\ddagger}$.

- Recall $\{\{\mathcal{B}_1,\mathcal{B}_2\}\} = \mathcal{B}^\ddagger$. Then, $\boxed{\{\{\mathcal{B}_1^\ddagger,\mathcal{B}_2^\ddagger\}\} = \mathcal{B}^\ddagger}$.

- **Base:** CHOICE, such that $\mathcal{B}^\ddagger = \mathcal{B}_i$ and $1 \le i \le 2$.
  Recall $1 \le i \le 2$. Then:
  - **Case:** $i = 1$.
    By REFL, $\mathcal{B}_1 \sqsupseteq \mathcal{B}_1$. Then, $\mathcal{B}_1 \sqsupseteq \mathcal{B}_i$. Then, $\boxed{\mathcal{B}_1 \sqsupseteq \mathcal{B}^\ddagger}$.
  - **Case:** $i = 2$. Similar to case $i = 1$.
- **Step:** TRANS, such that $\{\{\mathcal{B}_1,\mathcal{B}_2\}\} \sqsupseteq \mathcal{B}^\dagger \sqsupseteq \mathcal{B}^\ddagger$, for some $\mathcal{B}^\dagger$.
  - Recall $\{\{\mathcal{B}_1,\mathcal{B}_2\}\} \sqsupseteq \mathcal{B}^\dagger$. Then, by induction:
    * **Case 1:** $\mathcal{B}_1 \sqsupseteq \mathcal{B}_1^\dagger$ and $\mathcal{B}_2 \sqsupseteq \mathcal{B}_2^\dagger$ and $\{\mathcal{B}_1^\dagger,\mathcal{B}_2^\dagger\} = \mathcal{B}$, for some $\mathcal{B}_1^\dagger,\mathcal{B}_2^\dagger$.
      Recall $\mathcal{B}^\dagger \sqsupseteq \mathcal{B}^\ddagger$. Then, $\{\mathcal{B}_1^\dagger,\mathcal{B}_2^\dagger\} \sqsupseteq \mathcal{B}^\ddagger$. Then, by induction:
      · **Case 1a:** $\mathcal{B}_1^\dagger \sqsupseteq \mathcal{B}_1^\ddagger$ and $\mathcal{B}_2^\dagger \sqsupseteq \mathcal{B}_2^\ddagger$ and $\boxed{\{\mathcal{B}_1^\ddagger,\mathcal{B}_2^\ddagger\} = \mathcal{B}^\ddagger}$, $\boxed{\text{for some } \mathcal{B}_1^\ddagger,\mathcal{B}_2^\ddagger}$.
      Recall $\mathcal{B}_1 \sqsupseteq \mathcal{B}_1^\dagger \sqsupseteq \mathcal{B}_1^\ddagger$ and $\mathcal{B}_2 \sqsupseteq \mathcal{B}_2^\dagger \sqsupseteq \mathcal{B}_2^\ddagger$. Then, by TRANS, $\mathcal{B}_1 \sqsupseteq \mathcal{B}_1^\ddagger$ and $\mathcal{B}_2 \sqsupseteq \mathcal{B}_2^\ddagger$.
      · **Case 1b:** $\mathcal{B}_1^\dagger \sqsupseteq \mathcal{B}^\ddagger$.
      Recall $\mathcal{B}_1 \sqsupseteq \mathcal{B}_1^\dagger \sqsupseteq \mathcal{B}^\ddagger$. Then, by TRANS, $\boxed{\mathcal{B}_1 \sqsupseteq \mathcal{B}^\ddagger}$.
      · **Case 1c:** $\mathcal{B}_2^\dagger \sqsupseteq \mathcal{B}^\ddagger$. Similar to case 1b.
    * **Case 2:** $\mathcal{B}_1 \sqsupseteq \mathcal{B}^\dagger$.
      Recall $\mathcal{B}_1 \sqsupseteq \mathcal{B}^\dagger \sqsupseteq \mathcal{B}^\ddagger$. Then, by TRANS, $\boxed{\mathcal{B}_1 \sqsupseteq \mathcal{B}^\ddagger}$.
    * **Case 3:** $\mathcal{B}_2 \sqsupseteq \mathcal{B}^\dagger$. Similar to case 2.
- **Step:** CONGR, such that $\boxed{\mathcal{B}^\ddagger = \{\{\mathcal{B}_1^\ddagger,\mathcal{B}_2^\ddagger\}\}}$ and $\boxed{\mathcal{B}_1 \sqsupseteq \mathcal{B}_1^\ddagger}$ and $\boxed{\mathcal{B}_2 \sqsupseteq \mathcal{B}_2^\ddagger}$, $\boxed{\text{for some } \mathcal{B}_1^\ddagger,\mathcal{B}_2^\ddagger}$. □

Lemma 8. *Let R_1,R_2 be branching pomsets.*

(i) If $R_1 \sqsupseteq R_1'$ and $R_2 \sqsupseteq R_2'$ then $R_1 \parallel R_2 \sqsupseteq R_1' \parallel R_2'$.

(ii) If $R_1 \sqsupseteq R_1'$ and $R_2 \sqsupseteq R_2'$ then $R_1 + R_2 \sqsupseteq R_1'$, $R_1 + R_2 \sqsupseteq R_2'$ and $R_1 + R_2 \sqsupseteq R_1' + R_2'$.

(iii) If $R_1 \sqsupseteq R_1'$ and $R_2 \sqsupseteq R_2'$ then $R_1 ; R_2 \sqsupseteq R_1' ; R_2'$.

(iv) If $R_1 \dagger R_2 \sqsupseteq R_3$ for $\dagger \in \{;,\parallel\}$ then $R_3 = R_1' \dagger R_2'$ for some $R_1 \sqsupseteq R_1'$ and $R_2 \sqsupseteq R_2'$.

(v) If $R_1 + R_2 \sqsupseteq R_3$ then either $R_3 = R_1'$ or $R_3 = R_2'$ or $R_3 = R_1' + R_2'$ for some $R_1 \sqsupseteq R_1', R_2 \sqsupseteq R_2'$.

Proof. Let $R_1 = \langle E_1, \le_1, \lambda_1, \mathcal{B}_1\rangle$ with $\le_1 = \le_1^\star$ and similarly for R_2. By the rules in Fig. 7 $R_1' = R_1[\mathcal{B}_1'] = \langle E_1', \le_1', \lambda_1', \mathcal{B}_1'\rangle$ for some $\mathcal{B}_1 \sqsupseteq \mathcal{B}_1'$ and analogously for R_2'.

(i) By the rules in Fig. 9 $R_1 \parallel R_2 = \langle E_1 \cup E_2, \le_1 \cup \le_2, \lambda_1 \cup \lambda_2, \mathcal{B}_1 \cup \mathcal{B}_2\rangle$. By Lemma 7(i) $\mathcal{B}_1 \cup \mathcal{B}_2 \sqsupseteq \mathcal{B}_1' \cup \mathcal{B}_2 \sqsupseteq \mathcal{B}_1' \cup \mathcal{B}_2'$. It follows that $R_1 \parallel R_2 \sqsupseteq (R_1 \parallel R_2)[\mathcal{B}_1' \cup \mathcal{B}_2'] = \langle E_1' \cup E_2', \le_1' \cup \le_2', \lambda_1' \cup \lambda_2', \mathcal{B}_1' \cup \mathcal{B}_2'\rangle = R_1' \parallel R_2'$.

(ii) By the rules in Fig. 9 $R_1 + R_2 = \langle E_1 \cup E_2, \le_1 \cup \le_2, \lambda_1 \cup \lambda_2, \{\{\mathcal{B}_1,\mathcal{B}_2\}\}\rangle$. By the rules in Fig. 7 $\{\{\mathcal{B}_1,\mathcal{B}_2\}\} \sqsupseteq \mathcal{B}_1'$. It follows that $R_1 + R_2 \sqsupseteq (R_1 + R_2)[\mathcal{B}_1'] = \langle E_1', \le_1', \lambda_1', \mathcal{B}_1'\rangle = R_1'$. The case for R_2' is analogous. By the rules in Fig. 7 $\{\{\mathcal{B}_1,\mathcal{B}_2\}\} \sqsupseteq \{\{\mathcal{B}_1',\mathcal{B}_2'\}\}$. It follows that $R_1 + R_2 \sqsupseteq (R_1 + R_2)[\{\{\mathcal{B}_1',\mathcal{B}_2'\}\}] = R_1' + R_2'$.

(iii) By the rules in Fig. 7 $R_1 ; R_2 = \langle E_1 \cup E_2, \le_1 \cup \le_2 \cup \bigcup_{\mathsf{a} \in \mathcal{A}} E_{1_\mathsf{a}} \times E_{2_\mathsf{a}}, \lambda_1 \cup \lambda_2, \mathcal{B}_1 \cup \mathcal{B}_2\rangle$. By Lemma 7(i) $\mathcal{B}_1 \cup \mathcal{B}_2 \sqsupseteq \mathcal{B}_1' \cup \mathcal{B}_2 \sqsupseteq \mathcal{B}_1' \cup \mathcal{B}_2'$. It follows that $R_1 ; R_2 \sqsupseteq (R_1 ; R_2)[\mathcal{B}_1' \cup \mathcal{B}_2'] = \langle E_1' \cup E_2', \le_1' \cup \le_2' \cup \bigcup_{\mathsf{a} \in \mathcal{A}} E_{1_\mathsf{a}}' \times E_{2_\mathsf{a}}', \lambda_1' \cup \lambda_2', \mathcal{B}_1' \cup \mathcal{B}_2'\rangle = R_1' ; R_2'$.

(iv) By the rules in Fig. 7 $(R_1 \dagger R_2).\mathcal{B} \sqsupseteq \mathcal{B}'$ and $R_3 = (R_1 \dagger R_2)[\mathcal{B}']$ for some $\mathcal{B}'$. By the rules in Fig. 9 $(R_1 \dagger R_2).\mathcal{B} = R_1.\mathcal{B} \cup R_2.\mathcal{B}$. It follows from Lemma 7(ii) that $\mathcal{B}' = \mathcal{B}'_1 \cup \mathcal{B}'_2$ for some $R_1.\mathcal{B} \sqsupseteq \mathcal{B}'_1$ and $R_2.\mathcal{B} \sqsupseteq \mathcal{B}'_2$. By the rules in Fig. 7 $R_1 \sqsupseteq R_1[\mathcal{B}'_1] = R'_1$ and $R_2 \sqsupseteq R_2[\mathcal{B}'_2] = R'_2$. Finally, by the rules in Fig. 9 $R_3 = R'_1 \dagger R'_2$.

(v) By the rules in Fig. 7 $(R_1 + R_2).\mathcal{B} \sqsupseteq \mathcal{B}'$ and $R_3 = (R_1 + R_2)[\mathcal{B}']$ for some $\mathcal{B}'$. By the rules in Fig. 9 $(R_1 + R_2).\mathcal{B} = \{\{\mathcal{B}_1, \mathcal{B}_2\}\}$. It follows from Lemma 7(iii) that either $\mathcal{B}' = \{\{\mathcal{B}'_1, \mathcal{B}'_2\}\}$ or $\mathcal{B}' = \mathcal{B}'_1$ or $\mathcal{B}' = \mathcal{B}'_2$ for some $R_1.\mathcal{B} \sqsupseteq \mathcal{B}'_1, R_2.\mathcal{B} \sqsupseteq \mathcal{B}'_2$. By the rules in Fig. 7 $R_1 \sqsupseteq R_1[\mathcal{B}'_1] = R'_1$ and $R_2 \sqsupseteq R_2[\mathcal{B}'_2] = R'_2$. If $\mathcal{B}' = \{\{\mathcal{B}'_1, \mathcal{B}'_2\}\}$ then by the rules in Fig. 9 $R_3 = R'_1 + R'_2$. The other two cases are analogous. □

Lemma 9. *Let R_1, R_2 be branching pomsets. Let e be an event.*

(i) If $R_1 \xrightarrow{\checkmark_e} R'_1$ then $R_1 \parallel R_2 \xrightarrow{\checkmark_e} R'_1 \parallel R_2$.

(ii) If $R_1 \xrightarrow{\checkmark_e} R'_1$ then $R_1 + R_2 \xrightarrow{\checkmark_e} R'_1$.

(iii) If $R_1 \xrightarrow{\checkmark_e} R'_1$ then $R_1 ; R_2 \xrightarrow{\checkmark_e} R'_1 ; R_2$.

Proof.

(i) By Lemma 8(i) $R_1 \parallel R_2 \sqsupseteq R'_1 \parallel R_2$. Since $R_1 \xrightarrow{\checkmark_e} R'_1$, $e \in \text{a-min}(R'_1)$ and then $e \in \text{a-min}(R'_1 \parallel R_2)$.
Suppose that there exists some R'' such that $R_1 \parallel R_2 \sqsupseteq R'' \sqsupset R'_1 \parallel R_2$ and $e \in \text{a-min}(R'')$. Then $R'' = R''_1 \parallel R_2$ for some $R_1 \sqsupseteq R''_1 \sqsupset R'_1$ such that $e \in \text{a-min}(R''_1)$, but this contradicts our premise that $R_1 \xrightarrow{\checkmark_e} R'_1$. We conclude that there exists no such R'' and then by the rules in Fig. 7 $R_1 \parallel R_2 \xrightarrow{\checkmark_e} R'_1 \parallel R_2$.

(ii) By Lemma 8(ii) $R_1 + R_2 \sqsupseteq R'_1$. Since $R_1 \xrightarrow{\checkmark_e} R'_1$, $e \in \text{a-min}(R'_1)$.
Suppose that there exists some R'' such that $R_1 + R_2 \sqsupseteq R'' \sqsupset R'_1$ and $e \in \text{a-min}(R'')$. For the latter to be true we have to resolve the outer choice $R_1 + R_2$, so $R_1 \sqsupseteq R'' \sqsupset R'_1$, but this contradicts our premise that $R_1 \xrightarrow{\checkmark_e} R'_1$. We conclude that there exists no such R'' and then by the rules in Fig. 7 $R_1 + R_2 \xrightarrow{\checkmark_e} R'_1$.

(iii) By Lemma 8(iii) $R_1 ; R_2 \sqsupseteq R'_1 ; R_2$. Since $R_1 \xrightarrow{\checkmark_e} R'_1$, $e \in \text{a-min}(R'_1)$ and then $e \in \text{a-min}(R'_1 ; R_2)$.
Suppose that there exists some R'' such that $R_1 ; R_2 \sqsupseteq R'' \sqsupset R'_1 ; R_2$ such that $e \in \text{a-min}(R'')$. Then $R'' = R''_1 ; R_2$ for some $R_1 \sqsupseteq R''_1 \sqsupset R'_1$ such that $e \in \text{a-min}(R''_1)$, but this contradicts our premise that $R_1 \xrightarrow{\checkmark_e} R'_1$. We conclude that there exists no such R'' and then by the rules in Fig. 7 $R_1 ; R_2 \xrightarrow{\checkmark_e} R'_1 ; R_2$. □

Lemma 10. *Let c^* be a dependently guarded choreograpy and let $[\![c^*]\!] \xrightarrow{\ell} R'$ for some R'. Then $[\![c]\!] \xrightarrow{\ell} R''$ and $R' = R'' ; [\![c^*]\!]$ for some R''.*

Proof. Let $R = [\![c^*]\!]$. By Fig. 9 $R = [\![(R_1 ; R_2) + \mathbf{0}]\!]$ where $R_1 = [\![c]\!]$ and $R_2 = [\![c^*]\!]$. By Fig. 7 $R \xrightarrow{\checkmark_e} R_3$ such that $\lambda(e) = \ell$ and $R' = R_3 - e$. It follows that $R_1 ; R_2 \xrightarrow{\checkmark_e} R_3$. By Fig. 7 $R_1 ; R_2 \sqsupseteq R_3$ and then by Lemma 8(iv) $R_3 = R'_1 ; R'_2$ for some $R_1 \sqsupseteq R'_1$ and $R_2 \sqsupseteq R'_2$, and either $e \in \text{a-min}(R'_1)$ or $e \in \text{a-min}(R'_2)$. If $e \in \text{a-min}(R'_1)$ then $R_1 \xrightarrow{\checkmark_e} R'_1$ and by Lemma 9(iii) $R_1 ; R_2 \xrightarrow{\checkmark_e} R'_1 ; R_2$. It follows that $R_3 = R'_1 ; R_2$ and then $R' = (R'_1 - e) ; [\![c^*]\!]$. Otherwise, i.e. if $e \in \text{a-min}(R'_2)$, then $R_2 \xrightarrow{\checkmark_e} R'_2$ and by Lemma 1(b) $c \xrightarrow{\checkmark_\ell}$. However, since c^* is dependently guarded it follows that the subject of ℓ does not occur in c and then it also does not occur in c^*. As this is contradictory, e cannot be an event in R_2. □