Delayed-choice semantics for pomset families and message sequence graphs*

Author's Version – August 9, 2017

Clemens Dubslaff and Christel Baier

Faculty of Computer Science
Technische Universität Dresden
Dresden, Germany
{clemens.dubslaff,christel.baier}@tu-dresden.de

Abstract. Message sequence charts (MSCs) are diagrams widely used to describe communication scenarios. Their higher-order formalism is provided by graphs over MSCs, called message sequence graphs (MSGs), which naturally induce a non-interleaving linear-time semantics in terms of a pomset family. Besides this pomset semantics, an operational semantics for MSGs was standardized by the ITU-T as an interleaving branching-time semantics using a process-algebraic approach. A key ingredient in the latter semantics is *delayed choice*, formalizing that choices between communication scenarios are only made when they are inevitable. In this paper, an approach towards branching-time semantics for pomset families that follows the concept of delayed choice is proposed. First, *transition-system semantics* are provided where global states comprise cuts of pomsets represented either by suffixes or prefixes of family members. Second, an *event-structure semantics* is presented those benefit is to maintain the causal dependencies of events provided by the pomset family. These semantics are also investigated in the context of pomset families generated by MSGs.

1 Introduction

During the last decades, much effort has been put into developing models for concurrent systems to specify and reason about communication protocols. *Message sequence charts (MSCs)* provide an intuitive formalism to describe scenarios for asynchronously communicating processes. They are standardized by the ITU [20] and have been also included into the current UML 2.0 specification [13] as *sequence diagrams*. An MSC comprises time lines for each process on which events of the respective process are totally ordered, and message arrows that connect corresponding send and receive events between the processes. Two communication scenarios given by MSCs can be composed by extending the time

* The authors are supported by Deutsche Telekom Stiftung, by the DFG through the Collaborative Research Center SFB 912 – HAEC, the Excellence Initiative by the German Federal and State Governments (cluster of excellence cfAED), the DFG-projects BA-1679/11-1 and BA-1679/12-1, and the 5G Lab Germany.

lines of the first MSC by the time lines of the same process in the second MSC. This naturally allows for specifying collections of MSCs by graphs over MSCs, called *message sequence graphs (MSGs)*: an MSG describes all those MSCs that arise from sequential compositions of MSCs along paths in the MSG.

Example 1. In Figure 1, an example MSG is depicted that contains two non-empty MSCs, both describing a communication scenario where process s sends data to process r and waits for an acknowledgment. The left scenario branch models the case where a failure during the transmission of data occurred, imposing a timeout at process s. Then, s resends the data to r, being either unsuccessful again (see the self loop at the left box) or being successful (switching to the right box). In the latter case, when r successfully received all data, an acknowledgement is sent to s and the communication between s and r terminates.

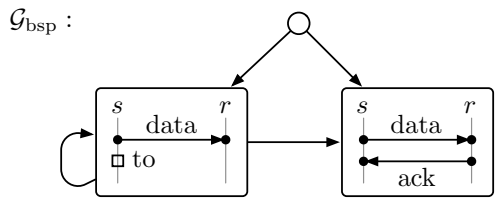


Fig. 1. An example MSG for sender-receiver communication scenarios

Semantics for MSGs. In the taxonomy of models for concurrent systems [36], different abstraction levels of semantics are provided along two orthogonal axes: one distinguishes between non-interleaving and interleaving and the other between branching-time and linear-time semantics. Non-interleaving semantics model causal dependencies and the independence of actions in an explicit way. Within branching-time semantics, the information when choices are made during an execution of the system can be modeled, while linear-time semantics abstract away from the points where these choices are made. To this end, non-interleaving branching-time semantics can be seen as the most general semantics in this taxonomy and interleaving linear-time semantics as the most abstracted ones.

MSCs naturally induce a non-interleaving linear-time semantics in terms of a *partially-ordered multiset (pomset)* [32], provided by the total time-line orderings and the fact that any send event has to precede its corresponding receive event [1]. The composition of MSCs is defined syntactically by gluing process time lines together. This corresponds to the *local concatenation* of pomsets [32] where events of the same process are assumed to depend on each other. Thus, a non-interleaving linear-time semantics for MSGs is naturally defined by a family of pomsets comprising the MSCs that arise from concatenating pomsets of MSCs along paths in the MSG. This semantics for MSGs is widely accepted in the literature and often used to reason about MSGs (cf., e.g., [21,30,28,2,23,6] and surveys [29,11]).

Although the pomset semantics for MSGs is very natural, the standard semantics for MSGs specified by the international telecommunication union (ITU-T) in [19] is an interleaving branching-time semantics, defined through a process-algebraic approach. Besides historical reasons, the choice of this approach has been mainly motivated by providing an operational semantics for MSGs that allows to reason about the step-wise behavior of systems specified by MSGs [25,33]. The basic building blocks of the process-algebraic semantics are process terms for MSCs [24] defined over atomic actions, standard concatenation $\cdot$, and standard choice $+$. These process terms have an interleaving branching-time semantics. The standardized process-algebraic semantics for MSGs is obtained by a well-known standard transformation from automata theory, leading to a regular expression over process terms for MSCs [33]. Within this regular expression, special operators for concatenation, choice and recursion [26,19,25,33][1] are employed:

- Concatenation is provided by *weak sequential composition* borrowed from [35] those purpose is to transfer the local concatenation on pomsets [32] to the world of interleaving branching-time semantics. The definition of weak sequential composition is based on a *permission relation* that specifies how process terms can be influenced by firing actions not contained in the process term itself but in its context.
- The choice operator is given by *delayed choice* [3], an intrinsic linear-time operator where choices between MSC fragments are delayed until they are inevitable. According to [3], "the delayed-choice operator acts as a deterministic choice in the context of strong bisimulation by unifying process-algebra terms with a common prefix".
- Recursion is defined by adapting the recursion operator of [5] for MSGs with rules for weak sequential composition and delayed choice [33].

The main challenge within the process-algebraic approach towards an operational semantics for MSGs comes into play when interpreting recursion on process terms that involve weak sequential composition and delayed choice, as their definition requires negative premises in the operator-defining rules. It is well known that negative premises in operator-specifying rules impose several difficulties when aiming towards fixed points [12]. Reniers, one of the authors of the standardized process-algebraic semantics [19], noticed in his doctoral thesis [33] that "the definition of the permission relation on recursive equations is extremely difficult and no solution is found there yet" (see page 160 of [33]). Although [33] adapted the standardized semantics to avoid recursive equations, his process-algebraic semantics still contains negative premises in operator-specifying rules. Besides others, this raises the question whether an operational semantics for MSGs could be defined without sophisticated operators in terms of weak se-

[1] The cited papers focus on *high-level MSCs*, i.e., MSGs which in addition allow for a hierarchical structure and parallel composition. As high-level MSCs can be unfolded into an MSG [2] without changing their operational semantics, we disregard the parallel composition operator and focus solely on MSGs in this paper.

quential composition and delayed choice but where the main concepts of the standardized process-algebraic semantics are maintained.

Our contribution. Following the concept of delayed choice but avoiding sophisticated process-algebraic operators, we define branching-time semantics for pomset families. During the presentation of our semantics, we exemplify their application to MSGs using the running example from the introduction. We first aim towards interleaving branching-time semantics (as within the process-algebraic approach defining delayed choice [3]) and present two different views on defining transition systems for pomset families:

(1) Suffix transition systems arise from interpreting delayed choice on pomset families that, similar to the process-algebraic approach, contain all possible future behaviors of pomset-family members. This semantics is used as a reference model for all other semantics in this paper.

(2) Prefix transition systems interpret delayed choice based on the past behaviors of the pomset family members. They are deterministic and bisimilar to the corresponding suffix transition system (1).

The purpose of suffix transition systems (1) is to closely follow the process-algebraic approach for defining delayed-choice semantics in the context of pomset families. Prefix transition systems (2) complete the picture of our approach and extend the delayed-choice semantics for MSGs provided in [10] towards arbitrary pomset families. This semantics thus provides the connection of the semantics defined throughout this paper to the approach by [10]. Every MSG naturally induces a pomset family where each member is an MSC. Thus, suffix and prefix transition systems also provide delayed-choice semantics for MSGs.

We furthermore present a non-interleaving branching-time semantics for pomset families in terms of a prime event structure [31]:

(3) Pomset event structures maintain the causality information between the events in the pomset family members. For pomsets generated by MSGs, the transition system induced by the pomset event structure is isomorphic to the corresponding prefix transition system (2).

As far as we know, pomset event structures (3) provide the first non-interleaving branching-time semantics for MSGs that follows the concept of delayed choice.

As intended by the authors of [35], it is quite natural that the weak sequential composition operator corresponds to the local concatenation of pomsets in our setting, not requiring the sophisticated definition of the permission relation that imposes difficulties within recursion. We show that the process-algebraic delayed-choice operator corresponds to the standard union operation in the setting of pomset families.

Further related work. The process-algebraic approach for the standard operational semantics for MSGs [26,19,25,33] uses interleaved models of MSCs as

basic building blocks. Based on these process terms for MSCs, sophisticated operators of weak sequential composition and delayed choice are used to mimic the linear-time behavior of MSGs in the branching-time setting. This is in contrast to our approach, where we use the pomset semantics for MSCs as basic building blocks and exploit standard linear-time operators *before* interleaving pomset families towards a branching-time semantics.

Besides using the process-algebraic approach, a transition-system semantics for MSGs has been presented in [9] also based on the pomset semantics for MSGs but not obeying delayed choice in the sense as we establish in this paper. In particular, the transition system of [9] is not deterministic. Most related to our pomset event structure (3) is the prime-event-structure semantics for MSGs presented in [14]. However, their recursive definition employs the standard choice operator instead of delayed choice, inducing a transition-system semantics that is non-deterministic and infinitely branching. Further branching-time semantics not following the concept of delayed choice have been presented as graphs with synchronization points [23], or Petri-net components [16].

In order to reason about quantitative aspects of MSGs, our previous work in [10] presented a transition-system semantics for MSGs that essentially corresponds to the prefix transition-system semantics (cf. (2) above).

2 Preliminaries

In this section, we recall basic concepts of models for concurrency, communication systems and notations used throughout this paper. By $\mathbb{N}$ we denote the set of non-negative integers. For any set X, we denote by 2^X the power set of X.

2.1 Models for concurrency

The models we use throughout this paper mainly follow the taxonomy detailed in the introduction (cf., e.g., [36]), comprising linear-time models in terms of *formal languages* (interleaving) and *pomsets* (non-interleaving), and branching-time models in terms of *transition systems* (interleaving) and *event structures* (non-interleaving). Furthermore, we follow the principle of atomic *actions*, i.e., tasks of the system indivisible on the abstraction level of the model, where we denote by Σ the alphabet of all actions. Instances of actions are *events* from a set E that are labeled by its corresponding action name via a labeling function $\lambda\colon E \to \Sigma$.

Formal languages. We denote by $\Sigma^\star$ and Σ^+ the set of all finite and non-empty finite words over Σ, respectively. By ε we denote the empty word. A *language* over Σ is a subset of $\Sigma^\star$. By $w[i]$ we denote the $(i{+}1)$-st symbol of a word w and by $|w|$ the length of w.

Pomsets. A labeled partial order is a tuple $\mathcal{P} = (E, \leqslant, \lambda)$, where $\leqslant$ is a partial order over a set of events E and $\lambda\colon E \to \Sigma$ is the labeling function. We identify isomorphic labeled partial orders, i.e., we treat them as *pomsets* [32]. We call a

pomset *basic* if λ is injective and *finite* if E is a finite set. If not stated differently, we assume any pomset to be finite. Furthermore, we restrict ourselves to pomsets that are not autoconcurrent, i.e., for all $e, e' \in E$ with $\lambda(e) = \lambda(e')$ we have either $e \leqslant e'$ or $e \geqslant e'$. The empty pomset is denoted by $\oslash$. The set of pomsets over Σ is denoted by $\mathbb{POS}_\Sigma$. A linearization of $\mathcal{P}$ is a word $w \in \Sigma^\star$ for which there is a bijection $\xi\colon \{0, ..., |w|-1\} \to E$ with $\lambda\big(\xi(i)\big) = w[i]$ and $\xi(i) \not> \xi(j)$ for all $i \leqslant j < |w|$. $\mathcal{P}$ is *total* if for all $e, e' \in E$ with $e \neq e'$ we have either $e < e'$ or $e > e'$. With abuse of notations, we identify total orders with their linearization. For a set of events F, we denote by $\mathcal{P}|_F$ the projection of $\mathcal{P}$ onto F, i.e., the pomset $(E \cap F, \leqslant_F, \lambda_F)$, where $e \leqslant_F e'$ iff $e, e' \in E \cap F$ and $e \leqslant e'$, and where $\lambda_F(e) = \alpha$ iff $e \in F \cap E$ and $\lambda(e) = \alpha$. The *upward closure* on F is defined as $\uparrow F = \{e \in E\colon \exists e' \in F. e \geqslant e'\}$. A pomset $\mathcal{P}'$ for which there is an upward closed F with $\mathcal{P}' = \mathcal{P}|_F$ is called *suffix* of $\mathcal{P}$. $\mathcal{P}'$ is an *α-suffix* of $\mathcal{P}$ if furthermore $E \backslash F = \{e\}$ with $\lambda(e) = \alpha$. The set of all (α-) suffixes of $\mathcal{P}$ is denoted by $\mathrm{Suff}(\mathcal{P})$ ($\mathrm{Suff}_\alpha(\mathcal{P})$, respectively). Accordingly, we define the notions of *downward closure* $\downarrow F$, and *prefixes* $\mathrm{Pref}(\mathcal{P})$ and $\mathrm{Pref}_\alpha(\mathcal{P})$. A *pomset family* is a set of pomsets, where we denote the set of all pomset families over Σ by $\mathbb{POF}_\Sigma = 2^{\mathbb{POS}_\Sigma}$. Notations for pomsets defined above extend to pomset families $\mathfrak{P} \in \mathbb{POF}_\Sigma$ as expected, e.g., for some $\alpha \in \Sigma$, $\mathrm{Suff}_\alpha(\mathfrak{P}) = \bigcup_{\mathcal{P} \in \mathfrak{P}} \mathrm{Suff}_\alpha(\mathcal{P})$. The notion of α-suffixes is generalized to pomset families $\mathrm{Suff}_w(\mathfrak{P})$ of w-suffixes of $\mathfrak{P}$ for $w \in \Sigma^\star$ by defining $\mathrm{Suff}_\varepsilon(\mathfrak{P}) = \mathfrak{P}$ and $\mathrm{Suff}_{w\alpha}(\mathfrak{P}) = \mathrm{Suff}_\alpha\big(\mathrm{Suff}_w(\mathfrak{P})\big)$, for $w \in \Sigma^\star$, $\alpha \in \Sigma$.

Transition systems. A *transition system* over an alphabet Σ is a tuple $\mathcal{T} = (S, \iota, \longrightarrow, \mathrm{Term})$ where S is a countable set of states, $\iota \in S$ an initial state, $\longrightarrow \subseteq S \times \Sigma \times S$ a transition relation, and $\mathrm{Term} \subseteq S$ a set of *termination states*. A *path* for a word $w = \alpha_0 \alpha_1 \cdots \alpha_{n-1}$ is a sequence $\pi = s_0 \alpha_0 s_1 \alpha_1 \ldots \alpha_{n-1} s_n$, where $s_0 = \iota$ and $s_i \xrightarrow{\alpha_i} s_{i+1}$ for all $i < n$. π is an *execution* for w if $s_n \in \mathrm{Term}$. $\mathcal{T}$ is *finite* if its reachable part $\{s \in S : \iota \longrightarrow^\star s\}$ is finite. $\mathcal{T}$ is *deterministic* if for all $s, t, t' \in S$ reachable in $\mathcal{T}$ and $\alpha \in \Sigma$ with $s \xrightarrow{\alpha} t$ and $s \xrightarrow{\alpha} t'$ we have $t = t'$. The set of all transition systems over Σ is denoted by $\mathbb{TS}_\Sigma$.

Event structures. As a branching-time non-interleaving model we rely on *(labeled) prime event structures* [31] and amend a notion of termination. A prime event structure is a tuple $\mathcal{E} = (E, \leqslant, \lambda, \#)$ that extends a possibly infinite pomset $(E, \leqslant, \lambda)$ with an irreflexive and symmetric *conflict relation* $\# \subseteq E \times E$ such that the following conditions hold:

(principle of finite causes) $\downarrow e$ is finite for all $e \in E$, and
(conflict heredity) $e \# e'$ and $e' \leqslant e''$ implies $e \# e''$ for all $e, e', e'' \in E$.

A *configuration* of $\mathcal{E}$ is a finite subset of events $C \subseteq E$ that is

downward-closed, i.e., $\downarrow C = C$, and
conflict-free, i.e., for all $e, e' \in C$ we never have $e \# e'$.

The set of configurations of $\mathcal{E}$ is denoted by $\mathrm{Conf}(\mathcal{E})$. A *prime event structure with termination* is a pair $(\mathcal{E}, \mathrm{Term})$, where $\mathcal{E}$ is a prime event structure and $\mathrm{Term} \subseteq \mathrm{Conf}(\mathcal{E})$ is a set of *termination configurations*. To simplify notations,

we use the notion of an *event structure* instead of "prime event structure with termination". We denote the set of all event structures over Σ by $\mathbb{ES}_\Sigma$.

2.2 Relations between models for concurrency

Following [36], we introduce mappings between the models of the last section:

- $\mathfrak{ts}\colon \mathbb{ES}_\Sigma \to \mathbb{TS}_\Sigma$ is the function that assigns the transition system $\mathfrak{ts}(\mathcal{E}) = \big(\mathrm{Conf}(\mathcal{E}), \varnothing, \longrightarrow, \mathrm{Term}\big)$ to an event structure $\mathcal{E} = (E, \leqslant, \lambda, \#, \mathrm{Term})$, where $C \xrightarrow{\alpha} D$ iff $D = C \cup \{e\}$ for some $e \in E$ with $\lambda(e) = \alpha$.
- $\mathfrak{pof}\colon \mathbb{ES}_\Sigma \to \mathbb{POF}_\Sigma$ is the function that assigns the pomset family $\mathfrak{pof}(\mathcal{E}) = \big\{(C, \leqslant|_C, \lambda|_C) : C \in \mathrm{Term}\big\}$ to an event structure $\mathcal{E} = (E, \leqslant, \lambda, \#, \mathrm{Term})$.
- $\mathfrak{lang}\colon \mathbb{TS}_\Sigma \to \Sigma^\star$ is the function that assigns the language $\mathfrak{lang}(\mathcal{T})$ to a transition system $\mathcal{T}$ comprising all words for which there is an execution in $\mathcal{T}$.
- $\mathfrak{lang}\colon \mathbb{POF}_\Sigma \to \Sigma^\star$ is the function that assigns the language $\mathfrak{lang}(\mathfrak{P}) = \bigcup_{\mathcal{P}\in\mathfrak{P}} \mathrm{Lin}(\mathcal{P})$ to a pomset family $\mathfrak{P}$, where $\mathrm{Lin}(\mathcal{P})$ denotes the set of linearizations of $\mathcal{P}$.[2]

It is well known that these functions commute, i.e., for any event structure $\mathcal{E}$ we have $\mathfrak{lang}\big(\mathfrak{ts}(\mathcal{E})\big) = \mathfrak{lang}\big(\mathfrak{pof}(\mathcal{E})\big)$.

Bisimulation. Bisimilarity is a central concept to compare the behavior of branching-time models [27]. A *bisimulation* between two transition systems $\mathcal{T} = (S, \iota, \longrightarrow, \mathrm{Term})$ and $\mathcal{T}' = (S', \iota', \longrightarrow', \mathrm{Term}')$ is a binary relation $\equiv \subseteq S \times S'$ where $\iota \equiv \iota'$,

(a) for all $s \in \mathrm{Term}$ there is an $s' \in \mathrm{Term}'$ with $s \equiv s'$, and
(a′) for all $s' \in \mathrm{Term}'$ there is an $s \in \mathrm{Term}$ with $s \equiv s'$,

and where for all $s \in S$ and $s' \in S'$ with $s \equiv s'$ we have that

(b) for all $t \in S$ with $s \xrightarrow{\alpha} t$ there is a $t' \in S'$ with $s' \xrightarrow{\alpha}{}' t'$ and $t \equiv t'$, and
(b′) for all $t' \in S'$ with $s' \xrightarrow{\alpha}{}' t'$ there is a $t \in S$ with $s \xrightarrow{\alpha} t$ and $t \equiv t'$.

If there exists a bisimulation between $\mathcal{T}$ and $\mathcal{T}'$, then $\mathcal{T}$ and $\mathcal{T}'$ are called *bisimilar*. A *bisimulation for* $\mathcal{T}$ is a bisimulation $\equiv$ between $\mathcal{T}$ and $\mathcal{T}$. We shall often use the well-known fact that bisimilarity coincides with trace equivalence for deterministic transition systems.

Lemma 1. *If $\mathcal{T}$ and $\mathcal{T}'$ are deterministic transition systems, then $\mathcal{T}$ and $\mathcal{T}'$ are bisimilar iff $\mathfrak{lang}(\mathcal{T}) = \mathfrak{lang}(\mathcal{T}')$.*

2.3 Modeling communication systems

Let P denote a finite set of *processes* and let Λ be a finite alphabet of *data labels*. To model communication between processes, we consider a special instance of the alphabet Σ. That is, we consider the set of communication actions $Act =$

[2] Note that we overload the function $\mathfrak{lang}$ for transition systems and pomset families.

$\bigcup_{p\in P} Act_p$, where Act_p comprises all actions a process $p \in P$ may perform, i.e., send actions $p!q(m)$ (process p sends a message m to process q), receive actions $p?q(m)$ (p receives m from q), and local actions $p(l)$ (p performs a local action l), for processes $q \in P$, and data labels $m, l \in \Lambda$. *Events* are instances of actions collected in a set E to which we assign actions by a labeling function $\lambda\colon E \to Act$. Given a set of events $F \subseteq E$, we denote by $F_!$ the set of send events, i.e., $F_! = \{e \in F : \exists p, q \in P, m \in \Lambda.\lambda(e) = p!q(m)\}$, and by $F_?$ the set of receive events, i.e., $F_? = \{e \in F : \exists p, q \in P, m \in \Lambda.\lambda(e) = p?q(m)\}$. Furthermore, for a process $p \in P$ and pomset $\mathcal{P} = (F, \leqslant, \lambda)$, we define $F_p = \{e \in F : \lambda(e) \in Act_p\}$.

Message sequence charts. The ITU-T standard [17] introduced *message sequence charts (MSCs)* as visual formalism for communication scenarios. Here, we recall the definition of MSCs based on pomsets [22].

Definition 1. *An* MSC *is a pomset $\mathcal{M} = (E, \leqslant, \lambda)$ for which $\mathcal{M}|_p$ is total for each $p \in P$ and where $\leqslant = \big(<_\mu \cup \bigcup_{p\in P} \leqslant|_{E_p}\big)^\star$. Here, $<_\mu = \big\{(e, \mu(e)) : e \in E_!\big\}$ denotes the binary relation defined for a bijection $\mu\colon E_! \to E_?$ with $\lambda(e) = p!q(m)$ and $\lambda(\mu(e)) = q?p(m)$ for all $e \in E_!$.*

The mapping μ in the above definition guarantees that the action labels are compatible with the interpretation of send and receive events, i.e., matches every receive event with a corresponding send event. The requirement that the events of a process are totally ordered formalizes the time-line ordering. We denote by $\mathbb{MSC}$ the set of all MSCs and by $\mathrm{b}\mathbb{MSC}$ the set of all basic MSCs, i.e., MSCs $(E, \leqslant, \lambda)$ where λ is injective.

Example 2. Let us return to the introductory Example 1 that models a simple set of communication scenarios over processes $P = \{s, r\}$. The MSC depicted on the left-hand side models a scenario with some timeout event, formalized by $\mathcal{M}_t = (E_t, \leqslant_t, \lambda_t)$ with $E_t = \{!, ?, to\}$, $\leqslant_t = \{(!, ?), (!, to)\}$, and $\lambda_t(!) = s!r(data)$, $\lambda_t(?) = r?s(data)$, and $\lambda_t(to) = s(to)$. Likewise, we formalize the MSC on the right-hand side by $\mathcal{M}_a = (E_a, \leqslant_a, \lambda_a)$, where $E_a = \{!d, ?d, !a, ?a\}$, $\leqslant_a = \big\{(!d, ?d), (!a, ?a), (?d, !a)\big\}^\star$, and where λ_a is given by $\lambda_a(!d) = s!r(data)$, $\lambda_a(?d) = r?s(data)$, $\lambda_a(!a) = r!s(ack)$, and $\lambda_a(?a) = s?r(ack)$. Note that both MSCs are basic.

Composition. Pomsets over Act are composed by performing a local concatenation where events of the same process depend on each other [32]. Formally, we define $\odot\colon \mathbb{P}\mathbb{O}_{Act} \times \mathbb{P}\mathbb{O}_{Act} \to \mathbb{P}\mathbb{O}_{Act}$ as follows: Let $\mathcal{X} = (X, \leqslant_X, \lambda_X)$ and $\mathcal{Y} = (Y, \leqslant_Y, \lambda_Y)$ be pomsets over Act with $X \cap Y = \varnothing$. Then, $\mathcal{X} \odot \mathcal{Y}$ is defined as the smallest pomset $\mathcal{Z} = (Z, \leqslant, \lambda)$ where $Z = X \cup Y$, $\leqslant|_X = \leqslant_X$, $\leqslant|_Y = \leqslant_Y$, and where for all $p \in P$, $e \in X_p$, $e' \in Y_p$ we have $e \leqslant e'$. The composition operation is extended to sequences of pomsets by $\odot\colon \mathbb{P}\mathbb{O}^\star_{Act} \to \mathbb{P}\mathbb{O}_{Act}$, where $\odot\varepsilon = \oslash$ and $\odot(\pi\mathcal{P}) = (\odot\pi) \odot \mathcal{P}$ for any $\pi \in \mathbb{P}\mathbb{O}^\star_{Act}$ and $\mathcal{P} \in \mathbb{P}\mathbb{O}_{Act}$. For a language over pomset sequences $L \subseteq \mathbb{P}\mathbb{O}^\star_{Act}$, we define $\odot L = \{\odot\pi : \pi \in L\}$.

In case the pomsets to be composed are MSCs, it is easy to see that composition again yields an MSC. Intuitively, composing MSCs corresponds to "gluing" their process lines together.

Message sequence graphs. Whereas MSCs model single communication scenarios, *message sequence graphs (MSGs)* were introduced in [18] as the standard higher-order formalism to specify collections of communication scenarios, i.e., sets of MSCs.

Definition 2. *An* MSG *is a tuple* $\mathcal{G} = (B, b_0, \hookrightarrow, \beta)$*, where* B *is a finite set of* boxes*,* $b_0 \in B$ *an initial box,* $\hookrightarrow \subseteq B \times B$ *a transition relation, and* $\beta\colon B \to \mathrm{b}\mathbb{MSC}$ *a labeling function that assigns basic MSCs to boxes.*

Note that we allow β for assigning the empty MSCs $\oslash$ to some box. We extend β towards $\beta\colon B^\star \to \mathbb{MSC}$ by inductively defining $\beta(\varepsilon) = \oslash$ and $\beta(\pi b) = \beta(\pi) \odot \beta(b)$ for $\pi \in B^\star$ and $b \in B$. A box sequence $\pi = b_0 b_1 \dots b_n \in B^\star$ is called a *path in* $\mathcal{G}$ if $b_i \hookrightarrow b_{i+1}$ for all $i < n$. If π cannot be prolonged in $\mathcal{G}$, i.e., there is no $b \in B$ such that $b_n \hookrightarrow b$, we call π an *execution* of $\mathcal{G}$ and define by $\mathcal{B}[\mathcal{G}]$ the *box language* of $\mathcal{G}$ as the set of executions of $\mathcal{G}$. In the following, we assume that any path in $\mathcal{G}$ can be prolonged towards an execution of $\mathcal{G}$. An MSC $\mathcal{M}$ is *accepted by* $\mathcal{G}$ if $\mathcal{M}$ arises from a composition along an execution of $\mathcal{G}$. The *pomset semantics* of $\mathcal{G}$ is the set of all MSCs accepted by $\mathcal{G}$, denoted by $\mathfrak{P}[\mathcal{G}] = \beta\big(\mathcal{B}[\mathcal{G}]\big)$.

Example 3. The the MSG $\mathcal{G}_{\mathrm{bsp}}$ from Example 1 is formalized by

$$\mathcal{G}_{\mathrm{bsp}} = \big(\{\iota, t, a\}, \iota, \{(\iota, t), (\iota, a), (t, t), (t, a)\}, \beta\big)$$

with $\beta(\iota) = \oslash$, $\beta(t) = \mathcal{M}_t$, and $\beta(a) = \mathcal{M}_a$ where $\mathcal{M}_t$ and $\mathcal{M}_a$ are the MSCs defined as in Example 2. The set of paths in $\mathcal{G}_{\mathrm{bsp}}$ is the regular language given by the regular expression $\varepsilon + \iota t^\star(a + \varepsilon)$. The box language $\mathcal{B}[\mathcal{G}_{\mathrm{bsp}}]$ is given by the regular expression $\iota t^\star a$. The communication scenario arising from the execution $\iota t a \in \mathcal{B}[\mathcal{G}_{\mathrm{bsp}}]$ is the MSC $(E, \leqslant, \lambda) = \beta(\iota) \odot \beta(t) \odot \beta(a) = \mathcal{M}_t \odot \mathcal{M}_a$, i.e., $E = E_t \cup E_a$, $\leqslant = \big(\leqslant_t \cup \leqslant_a \cup \{(to, !d), (?, ?d)\}\big)^\star$, and $\lambda = \lambda_t \cup \lambda_a$.

3 Transition systems for pomset families

In this section, we present transition systems for pomset families that follow the concept of delayed choice. For this, let us fix an alphabet of actions Σ. Although we do not consider process algebras in detail here, let us support the intuition behind delayed choice operator $\mp$ by providing the process-algebraic rules of [3], where $\mp$ has been defined first. A *process term* stands for a possible future behavior of the system, where reductions on the terms are made through firing actions. The operational semantics of $\mp$ is provided by the rules **(DC1)**, **(DC2)**, **(DC3)**, and symmetric rules with exchanged roles for process terms x and y (and process terms x' and y', respectively). In these rules, $x \xrightarrow{\alpha} y$ stands

$$\frac{\mathrm{term}(x)}{\mathrm{term}(x \mp y)}\ \textbf{(DC1)} \qquad \frac{x \xrightarrow{\alpha} x' \qquad y \not\xrightarrow{\alpha}}{(x \mp y) \xrightarrow{\alpha} x'}\ \textbf{(DC2)} \qquad \frac{x \xrightarrow{\alpha} x' \qquad y \xrightarrow{\alpha} y'}{(x \mp y) \xrightarrow{\alpha} (x' \mp y')}\ \textbf{(DC3)}$$

for an execution of an action $\alpha \in \Sigma$ in x, $x \not\xrightarrow{\alpha}$ expresses that action α is not

executed by x, and term(x) stands for x having the option to terminate. Hence, **(DC1)** states that $x \mp y$ can terminate if x can. The other rules illustrate exactly the intuitive behavior of delayed choice: the choice between two process terms x and y is not resolved when they perform the same actions (cf., **(DC3)**), but when from x an action is performed that is not enabled in y (cf., **(DC2)**).

Operational semantics for pomset families. To specify the operational behavior of systems described by some pomset family $\mathfrak{P}$, a central aspect is to identify the global state of the system and to describe the step-wise behavior from each of the states. For a single pomset $\mathcal{P} = (E, \leqslant, \lambda)$, a global state is traditionally defined by a *cut*, i.e., a partition of the events E into past events U and future events $V = E \backslash U$. Thus, every cut of $\mathcal{P}$ can be specified by either the set of past events U or future events V – the respective other set of events follows from the fixed set of events E. From a cut, action α can be performed when there is a minimal future event $e \in V$ labeled by α. After performing α, the event e from the set of future events V is moved to the set of past events U.

Having the interpretation of delayed choice in mind, where process terms describe future behaviors, we may describe cuts by their future events and amend the partial order and labeling inherited from the given pomset. This yields a formalization of the stepwise behavior $\overset{\alpha}{\rightharpoonup}$ of executing an action α over (future) pomsets $\mathcal{Y} \in \mathbb{POS}_\Sigma$ by $\mathcal{Y} \overset{\alpha}{\rightharpoonup} \mathrm{Suff}_\alpha(\mathcal{Y})$.[3] The principle of the operational behavior for single pomsets can be generalized towards pomset families by $\rightharpoonup \subseteq \mathbb{POF}_\Sigma \times \Sigma \times \mathbb{POF}_\Sigma$, where the standard union on pomset families serves as delayed choice within the step-wise behavior $\rightharpoonup$ described above. That is, for a pomset family $\mathfrak{X} \in \mathbb{POF}_\Sigma$, we have $\mathfrak{X} \overset{\alpha}{\rightharpoonup} \mathfrak{Y}$ iff $\mathfrak{Y} = \mathrm{Suff}_\alpha(\mathfrak{X})$. It is easy to check that the rules for delayed choice specified for process algebra terms are fulfilled by $\rightharpoonup$ (cf. **(DC2)** and **(DC3)**), replacing $\overset{\alpha}{\to}$ by $\overset{\alpha}{\rightharpoonup}$ and using pomset families instead of process terms. Here, $\mathfrak{X} \overset{\alpha}{\not\rightharpoonup}$ denotes that $\mathrm{Suff}_\alpha(\mathfrak{X}) = \varnothing$. Furthermore, the empty pomset $\oslash \in \mathfrak{X}$ naturally serves as candidate for term($\mathfrak{X}$) (cf. **(DC1)**).

Formalizing the approach with the step-wise operational behavior for pomset families described above, we obtain *suffix transition systems* where cuts of the pomset family members are represented by the pomset of future events. To complete the picture, we further present *prefix transition systems* for pomset families where cuts are represented by pomsets of past events.

For the remainder of this section, let us fix a pomset family $\mathfrak{P} \in \mathbb{POF}_\Sigma$.

3.1 Suffix transition systems

Definition 3. *The* suffix transition system *of* $\mathfrak{P}$ *is given by*

$$\mathcal{T}_{\mathrm{suff}}[\mathfrak{P}] \quad = \quad \big(2^{\mathrm{Suff}(\mathfrak{P})}, \mathfrak{P}, \rightharpoonup, \mathfrak{T}\big)$$

where $\mathfrak{T} = \{\mathfrak{X} \subseteq \mathrm{Suff}(\mathfrak{P}) : \oslash \in \mathfrak{X}\}$*, and where for* $\mathfrak{X}, \mathfrak{Y} \subseteq \mathrm{Suff}(\mathfrak{P})$ *we have* $\mathfrak{X} \overset{\alpha}{\rightharpoonup} \mathfrak{Y}$ *iff* $\mathfrak{Y} = \mathrm{Suff}_\alpha(\mathfrak{X})$.

[3] Note that $\mathrm{Suff}_\alpha(\mathcal{Y})$ is a singleton as we assume pomsets to be not autoconcurrent.

Note that states of the suffix transition system of $\mathfrak{P}$ might be an infinite pomset family in case $\mathfrak{P}$ is infinite.

Example 4. Let us return to our running example, i.e., let us consider the MSG $\mathcal{G}_{\text{bsp}}$ from Example 3. By the definition of the pomset semantics for MSGs, we have $\mathfrak{P}[\mathcal{G}_{\text{bsp}}] = \{\mathcal{M}_t \odot \ldots \odot \mathcal{M}_t \odot \mathcal{M}_a\}$. For illustration purposes, we extend the definition of $\odot$ towards pomset families by $\mathcal{X} \odot \mathcal{Y} = \{\mathcal{X} \odot \mathcal{Y} : \mathcal{Y} \in \mathcal{Y}\}$. Using this abbreviation and with $\mathfrak{P} = \mathfrak{P}[\mathcal{G}_{\text{bsp}}]$, Figure 2 shows a fragment of the suffix transition system $\mathcal{T}_{\text{suff}}[\mathfrak{P}]$ with initial state $\mathfrak{P}$ and one termination state $\{\oslash\}$. $\mathcal{T}_{\text{suff}}[\mathfrak{P}]$ contains cycles and is infinite since for all $k \in \mathbb{N}$ the pomset family

$$\underbrace{\mathcal{M}_t|_{\{?\}} \odot \ldots \odot \mathcal{M}_t|_{\{?\}}}_{k \text{ times}} \odot \big((\mathcal{M}_t|_{\{?,to\}} \odot \mathfrak{P}) \cup \{\mathcal{M}_a|_{\{?d,!a,?a\}}\}\big)$$

is a reachable state in $\mathcal{T}_{\text{suff}}[\mathfrak{P}]$.

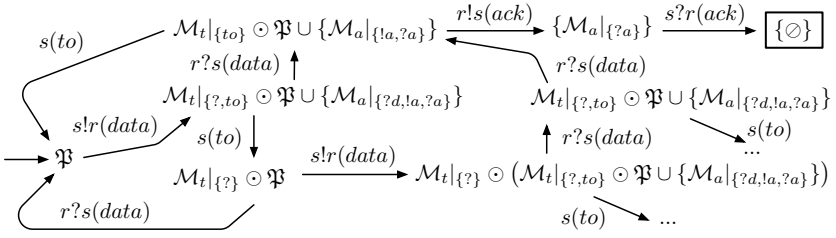


Fig. 2. Fragment of the suffix transition system for $\mathfrak{P}[\mathcal{G}_{\text{bsp}}]$

Proposition 1 (Properties of $\mathcal{T}_{\text{suff}}[\mathfrak{P}]$). *Given $\mathfrak{P} \in \mathbb{POF}_{\Sigma}$,*

(a) $\mathcal{T}_{\text{suff}}[\mathfrak{P}]$ *is deterministic, and*
(b) $\mathfrak{lang}\big(\mathcal{T}_{\text{suff}}[\mathfrak{P}]\big) = \mathfrak{lang}(\mathfrak{P})$.

Proof. It is easy to see by the definition of $\text{Suff}_{\alpha}(\cdot)$ that $\mathcal{T}_{\text{suff}}[\mathfrak{P}]$ is deterministic[4]. Let us now show language equality of $\mathcal{T}_{\text{suff}}[\mathfrak{P}]$ and $\mathfrak{P}$:

($\Rightarrow$) Let $w \in \mathfrak{lang}\big(\mathcal{T}_{\text{suff}}[\mathfrak{P}]\big)$, i.e., there is an execution $\mathfrak{X}_0\alpha_0\mathfrak{X}_1\alpha_1 \ldots \alpha_{n-1}\mathfrak{X}_n$ of $\mathcal{T}_{\text{suff}}[\mathfrak{P}]$ with $\mathfrak{X}_0 = \mathfrak{P}$, $\mathfrak{X}_n \in \mathfrak{T}$ and $w = \alpha_0\alpha_1 \ldots \alpha_{n-1}$. Then, by the definition of $\mathfrak{T}$, $\oslash \in \mathfrak{X}_n$. Furthermore, by the definition of $\text{Suff}_{\alpha}(\cdot)$, there is a $\mathcal{P} \in \mathfrak{X}_0$ such that for all $i < n$ there is $\mathcal{P}_i \in \mathfrak{X}_i$ with $\mathcal{P}_0 = \mathcal{P}$, $\mathcal{P}_n = \oslash$ and $\mathcal{P}_{i+1} = \text{Suff}_{\alpha_i}(\mathcal{P}_i)$. We show that w is a linearization of $\mathcal{P}_0 = (E, \leqslant, \lambda)$, i.e., there is a bijection $\xi\colon \{0, \ldots, n-1\} \to E$ with $\lambda\big(\xi(i)\big) = \alpha_i$ and $\xi(i) \not> \xi(j)$ for all $i \leqslant j < n$. For $i < n$, let $\xi(i)$ be the uniquely defined event in $\mathcal{P}_i \backslash \mathcal{P}_{i+1}$. Then, since $\mathcal{P}_{i+1} = \text{Suff}_{\alpha_i}(\mathcal{P}_i)$, $\lambda\big(\xi(i)\big) = \alpha_i$ for all $i < n$. Towards a contradiction, assume that there are $i \leqslant j$ such that $\xi(i) > \xi(j)$. Then, by the definition of suffixes, $\mathcal{P}_j$ is upward closed and thus, $\xi(i) \in \mathcal{P}_j$. Hence, by the definition of $\text{Suff}_{\alpha}(\cdot)$, for all $k \geqslant i$ we have $\xi(i) \in \mathcal{P}_k$, which yields $\xi(i) \notin \mathcal{P}_i \backslash \mathcal{P}_{i+1}$, contradicting the definition of ξ.

[4] Recall that determinism depends only on the reachable part in $\mathcal{T}_{\text{suff}}[\mathfrak{P}]$.

($\Leftarrow$) Let $w \in \mathfrak{lang}(\mathfrak{P})$, i.e., there is some $\mathcal{P} \in \mathfrak{P}$ such that $w \in \mathrm{Lin}(\mathcal{P})$. With $w = \alpha_0\alpha_1 \dots \alpha_{n-1}$ and $\mathcal{P} = (E, \leqslant, \lambda)$ there is thus a bijection $\xi\colon \{0, \dots, n{-}1\} \to E$ with $\lambda\big(\xi(i)\big) = \alpha_i$ and $\xi(i) \not> \xi(j)$ for all $i \leqslant j < n$. Let $\mathcal{P}_i$ for $i \leqslant n$ be inductively defined by $\mathcal{P}_0 = \mathcal{P}$ and $\mathcal{P}_{i+1} = \mathcal{P}_i \backslash \{\xi(i)\}$. Then for all $i < n$ we have that $\xi(i)$ is a minimal event in $\mathcal{P}_i$ and thus, by the definition of suffixes, $\mathcal{P}_{i+1} = \mathrm{Suff}_{\alpha_i}(\mathcal{P}_i)$. Thus, there is a path $\pi = \mathfrak{X}_0\alpha_0\mathfrak{X}_1\alpha_1 \dots \alpha_{n-1}\mathfrak{X}_n$ in $\mathcal{T}_{\mathrm{suff}}[\mathfrak{P}]$ with $\mathfrak{X}_0 = \mathfrak{P}$ and where $\mathcal{P}_i \in \mathfrak{X}_i$ for all $i \leqslant n$. As $\oslash = \mathcal{P}_n$, we have that $\oslash \in \mathfrak{X}_n$ and hence, π is an execution in $\mathcal{T}_{\mathrm{suff}}[\mathfrak{P}]$. This directly yields $w \in \mathfrak{lang}\big(\mathcal{T}_{\mathrm{suff}}[\mathfrak{P}]\big)$. □

Motivated by the last proposition and the fact that the step-wise behavior and the termination states satisfy the rules for delayed choice on suffix pomset families as illustrated in the introductory argumentation of this section, we use suffix transition systems as a reference model for delayed-choice semantics on pomset families.

Definition 4. *A transition system $\mathcal{T}$ for a pomset family $\mathfrak{P}$* obeys delayed choice *if $\mathcal{T}$ is deterministic and bisimilar to $\mathcal{T}_{\mathrm{suff}}[\mathfrak{P}]$.*

3.2 Prefix transition systems

We now define prefix transition systems, where states are given by prefixes of a pomset family. Intuitively, any prefix stands for the partially ordered history of an execution of the system. Prefix transition systems generalize the transition-system semantics for MSGs of [10] towards arbitrary pomset families (possibly not generated by MSGs).

Definition 5. *The* prefix transition system semantics *of $\mathfrak{P}$ is given by*

$$\mathcal{T}_{\mathrm{pref}}[\mathfrak{P}] \quad = \quad \big(2^{\mathrm{Pref}(\mathfrak{P})}, \{\oslash\}, \rightsquigarrow, \mathfrak{T}\big),$$

where $\mathfrak{T} = \{\mathfrak{X} \subseteq \mathrm{Pref}(\mathfrak{P}) : \mathfrak{X} \cap \mathfrak{P} \neq \oslash\}$ and where for $\mathfrak{X}, \mathfrak{Y} \subseteq \mathrm{Pref}(\mathfrak{P})$ we have $\mathfrak{X} \overset{\alpha}{\rightsquigarrow} \mathfrak{Y}$ iff $\mathfrak{X} = \mathrm{Pref}_\alpha(\mathfrak{Y})$.

Example 5. In Figure 3, a fragment of the prefix transition system $\mathcal{T}_{\mathrm{pref}}\big[\mathfrak{P}[\mathcal{G}_{\mathrm{bsp}}]\big]$ is depicted where $\mathcal{G}_{\mathrm{bsp}}$ is as in Example 3. Note that in this example, every reachable pomset family is a singleton.

Proposition 2 (Properties of $\mathcal{T}_{\mathrm{pref}}[\mathfrak{P}]$). *Given $\mathfrak{P} \in \mathbb{POF}_\Sigma$,*

(a) $\mathcal{T}_{\mathrm{pref}}[\mathfrak{P}]$ *is acyclic,*
(b) $\mathcal{T}_{\mathrm{pref}}[\mathfrak{P}]$ *is deterministic, and*
(c) $\mathfrak{lang}\big(\mathcal{T}_{\mathrm{pref}}[\mathfrak{P}]\big) = \mathfrak{lang}(\mathfrak{P})$.

Proof. In order to show that $\mathcal{T}_{\mathrm{pref}}[\mathfrak{P}]$ is acyclic, we rely on the fact that for all $\mathfrak{X} \subseteq \mathrm{Pref}(\mathfrak{P})$ reachable in $\mathcal{T}_{\mathrm{pref}}[\mathfrak{P}]$ we have that $\mathcal{X}, \mathcal{Y} \in \mathfrak{X}$ implies $|\mathcal{X}| = |\mathcal{Y}|$. Let now $\#(\mathfrak{X})$ denote the number of events contained in each $\mathcal{X} \in \mathfrak{X}$. Then, for all reachable $\mathfrak{X}, \mathfrak{Y} \subseteq \mathrm{Pref}(\mathfrak{P})$ with $\mathfrak{X} \overset{\alpha}{\rightsquigarrow} \mathfrak{Y}$ we have $\#(\mathfrak{Y}) = \#(\mathfrak{X}){+}1$. Thus, $\mathcal{T}_{\mathrm{pref}}[\mathfrak{P}]$ is acyclic. For any $\mathfrak{Y} \subseteq \mathrm{Pref}(\mathfrak{P})$ and $\alpha \in \Sigma$, $\mathfrak{X} = \mathrm{Pref}_\alpha(\mathfrak{Y})$ is uniquely defined and by the definition of $\overset{\alpha}{\rightsquigarrow}$, we directly obtain that $\mathcal{T}_{\mathrm{pref}}[\mathfrak{P}]$ is deterministic. Let us now show language equality of $\mathcal{T}_{\mathrm{pref}}[\mathfrak{P}]$ and $\mathfrak{P}$:

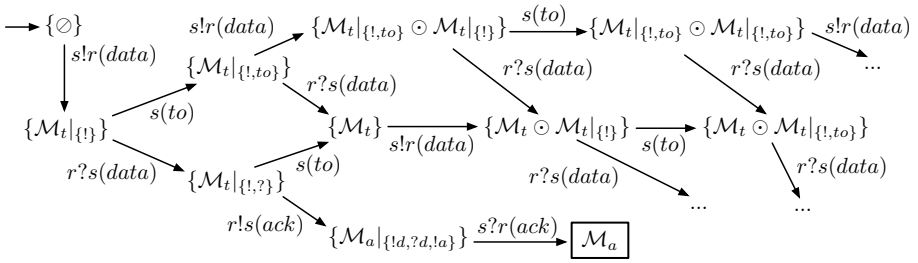


Fig. 3. Fragment of the prefix transition system for $\mathfrak{P}[\mathcal{G}_{\text{bsp}}]$

($\Rightarrow$) Let $w \in \mathfrak{lang}(\mathcal{T}_{\text{pref}}[\mathfrak{P}])$, i.e., there is an execution $\mathfrak{X}_0\alpha_0\mathfrak{X}_1\alpha_1\ldots\alpha_{n-1}\mathfrak{X}_n$ of $\mathcal{T}_{\text{pref}}[\mathfrak{P}]$ with $\mathfrak{X}_0 = \{\emptyset\}$, $\mathfrak{X}_n \in \mathfrak{T}$ and $w = \alpha_0\alpha_1\ldots\alpha_{n-1}$. Then, by the definition of $\mathfrak{T}$, there is some $\mathcal{P}_n = (E, \leqslant, \lambda) \in \mathfrak{X}_n$ such that $\mathcal{P}_n \in \mathfrak{P}$. Furthermore, by the definition of $\text{Pref}_\alpha(\cdot)$, for all $i < n$ there is $\mathcal{P}_i \in \mathfrak{X}_i$ with $\mathcal{P}_i = \text{Pref}_{\alpha_i}(\mathcal{P}_{i+1})$. We show that w is a linearization of $\mathcal{P}$, i.e., there is a bijection $\xi\colon \{0,\ldots,n{-}1\} \to E$ with $\lambda(\xi(i)) = \alpha_i$ and $\xi(i) \not> \xi(j)$ for all $i \leqslant j < n$. For $i < n$, let $\xi(i)$ be the uniquely defined event in $\mathcal{P}_{i+1}\backslash\mathcal{P}_i$. Then, since $\mathcal{P}_i = \text{Pref}_{\alpha_i}(\mathcal{P}_{i+1})$, $\lambda(\xi(i)) = \alpha_i$ for all $i < n$. Towards a contradiction, assume that there are $i \leqslant j$ such that $\xi(i) > \xi(j)$. Then, by the definition of prefixes, $\mathcal{P}_{i+1}$ is downward closed and thus, $\xi(j) \in \mathcal{P}_{i+1}$. Hence, by the definition of $\text{Pref}_\alpha(\cdot)$, for all $k > i$ we have $\xi(j) \in \mathcal{P}_k$, which yields $\xi(j) \notin \mathcal{P}_{j+1}\backslash\mathcal{P}_j$, contradicting the definition of ξ.

($\Leftarrow$) Let $w \in \mathfrak{lang}(\mathfrak{P})$, i.e., there is some $\mathcal{P} \in \mathfrak{P}$ such that $w \in \text{Lin}(\mathcal{P})$. With $w = \alpha_0\alpha_1\ldots\alpha_{n-1}$ and $\mathcal{P} = (E, \leqslant, \lambda)$ there is thus a bijection $\xi\colon \{0,\ldots,n{-}1\} \to E$ with $\lambda(\xi(i)) = \alpha_i$ and $\xi(i) \not> \xi(j)$ for all $i \leqslant j < n$. Let E_i for $i \leqslant n$ be inductively defined by $E = \emptyset$ and $E_{i+1} = E_i \cup \{\xi(i)\}$. Furthermore, let $\mathcal{P}_i = \mathcal{P}|_{E_i}$ for all $i \leqslant n$. Since $\xi(i) \not> \xi(j)$ for all $i \leqslant j < n$, all $\mathcal{P}_i$ are downward closed and thus, by the definition of prefixes, $\mathcal{P}_i = \text{Pref}_{\alpha_i}(\mathcal{P}_{i+1})$. Thus, there is a path $\pi = \mathfrak{X}_0\alpha_0\mathfrak{X}_1\alpha_1\ldots\alpha_{n-1}\mathfrak{X}_n$ in $\mathcal{T}_{\text{pref}}[\mathfrak{P}]$ with $\mathfrak{X}_0 = \{\emptyset\}$ and where $\mathcal{P}_i \in \mathfrak{X}_i$ for all $i \leqslant n$. As $\mathcal{P}_n = \mathcal{P}$, we have that $\mathcal{P} \in \mathfrak{X}_n$ and hence, $\mathfrak{X}_n \cap \mathfrak{P} \neq \emptyset$. Thus, π is an execution in $\mathcal{T}_{\text{pref}}[\mathfrak{P}]$ and hence, $w \in \mathfrak{lang}\big(\mathcal{T}_{\text{pref}}[\mathfrak{P}]\big)$. □

The above proposition in combination with Proposition 1 and Lemma 1 directly yields that $\mathcal{T}_{\text{pref}}[\mathfrak{P}]$ is a delayed-choice semantics for $\mathfrak{P}$:

Theorem 1. *$\mathcal{T}_{\text{pref}}[\mathfrak{P}]$ obeys delayed choice, i.e., $\mathcal{T}_{\text{pref}}[\mathfrak{P}]$ is deterministic and bisimilar to $\mathcal{T}_{\text{suff}}[\mathfrak{P}]$.*

3.3 Comparison and discussion

To further illustrate the differences between suffix and prefix transition systems, let us consider a simple example issuing a pomset family $\mathfrak{P} = \{\mathcal{X}, \mathcal{Y}\}$

that comprises the pomsets $\mathcal{X} = \big(\{e,e'\},\{(e,e')\},\{(e,\alpha),(e',\alpha')\}\big)$ and $\mathcal{Y} = \big(\{e,e'\},\varnothing,\{(e,\alpha),(e',\alpha')\}\big)$. Figure 4 depicts the resulting suffix and prefix transition systems for $\mathfrak{P}$. When executing α followed by α', the choice between $\mathcal{X}$ and $\mathcal{Y}$ is delayed, i.e., this execution could follow either $\mathcal{X}$ or $\mathcal{Y}$. However, when executing α' first, the choice between $\mathcal{X}$ and $\mathcal{Y}$ is resolved towards $\mathcal{Y}$. Whereas

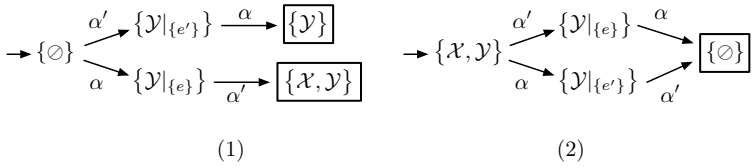


Fig. 4. The (1) prefix and (2) suffix transition system for $\mathfrak{P} = \{\mathcal{X},\mathcal{Y}\}$

in the case of the suffix transition system there is only one termination state, the prefix transition system contains the history of the execution and has two termination states. Note that both transition systems contain states which comprise more than one pomset.

Using the process algebra introduced in [19,33], we can describe $\mathfrak{P}$ by the process-algebraic term $(\alpha\|\alpha')\mp(\alpha{\cdot}\alpha')$. Using the rules specified in [33], we obtain the transition system depicted in (1) of Figure 5, which corresponds to the prefix transition system for $\mathfrak{P}$. Identifying bisimilar process-algebraic terms using the bisimulation $\underline{\leftrightarrow}$ provided in [33] yields a transition system corresponding to the suffix transition system for $\mathfrak{P}$, depicted in (2) of Figure 5.

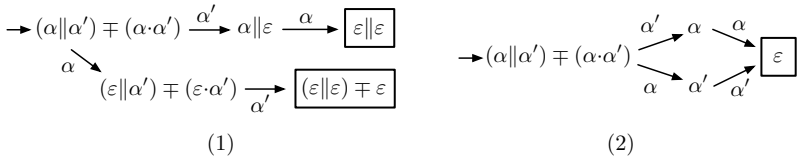


Fig. 5. Transition system of $(\alpha\|\alpha')\mp(\alpha{\cdot}\alpha')$ (1) and its quotient w.r.t. $\underline{\leftrightarrow}$ (2)

Note that in contrast to the prefix transition system $\mathcal{T}_{\text{pref}}\big[\mathfrak{P}[\mathcal{G}_{\text{bsp}}]\big]$ detailed in Example 5, $\mathcal{T}_{\text{pref}}[\mathfrak{P}]$ contains reachable states that are not singletons.

Lemma 2. *For all MSGs $\mathcal{G}$, the reachable states of $\mathcal{T}_{\text{pref}}\big[\mathfrak{P}[\mathcal{G}]\big]$ are singletons.*

Proof. As $\mathcal{T}_{\text{pref}}\big[\mathfrak{P}[\mathcal{G}]\big]$ is deterministic (see Proposition 2b), every action sequence $w \in Act^\star$ for which there is a path yields a uniquely defined state that we denote by $\mathfrak{X}_w$. Towards an induction on w, the statement holds for $w=\varepsilon$ as then, $\mathfrak{X}_w = \{\varnothing\}$. Let w be such that $\mathfrak{X}_w = \big\{(E,\leqslant,\lambda)\big\}$. Consider an $\alpha \in Act_p$

for a process $p \in P$ such that there is an α-transition in $\mathfrak{X}_w$ leading to $\mathfrak{X}_{w\alpha}$. Let $\mathcal{X} \in \mathfrak{X}_{w\alpha}$ with $\mathcal{X} = \big(E \cup \{e\}, \leqslant \cup (X \times \{e\}), \lambda \cup \{(e,\alpha)\}\big)$ for $X \subseteq E$. This can be assumed w.l.o.g. due to the definition of $\mathrm{Pref}_\alpha(\cdot)$. If α is a local or send event, then $X = {\downarrow}E_p \times \{e\}$ due to the definition of $\odot$ and the fact that local and send events have at most one direct predecessor in an MSC. Let $\alpha = p?q(m)$ and k denote the number of α-events in E. Since every receive event is mapped to a send event in a basic MSC, this mapping takes over to MSCs in $\mathfrak{P}[\mathcal{G}]$ by the definition of $\odot$. Thus, the kth event labeled by $q!p(m)$ on the process line E_q is a direct predecessor of e in $\mathcal{X}$. Since every receive event has at most two direct predecessors, we obtain $X = {\downarrow}\hat{e} \cup {\downarrow}E_p$ again by the definition of $\odot$. Hence, X is uniquely defined through $(E, \leqslant, \lambda)$ and α, leading to $\mathfrak{X}_{w\alpha}$ being a singleton. □

4 An event structure for pomset families

In this section, we present a branching-time semantics for pomset families that is non-interleaving, i.e., models causal dependencies and independence explicitly. Throughout this section, we fix a pomset family $\mathfrak{P}$ over Σ. Similar to concepts of [34], we define an event structure for $\mathfrak{P}$ where events are pomsets that arise from the downward closure of an event in some pomset of $\mathfrak{P}$. More formally, for a pomset $\mathcal{P} \in \mathfrak{P}$ with $\mathcal{P} = (F, \preceq, \nu)$ and $e \in F$, we consider the *pomset downward closure* of e as $\mathcal{P}|_{\downarrow e}$ with ${\downarrow}e = \{e' \in F : e' \preceq e\}$.

Definition 6. *The* pomset event structure $\mathcal{E}[\mathfrak{P}]$ *is given by* $(E, \leqslant, \lambda, \#, \mathrm{Term})$ *where*

- $E = \big\{\mathcal{P}|_{\downarrow e} : \mathcal{P} = (F, \preceq, \nu) \in \mathfrak{P}, e \in F\big\}$
- $\mathcal{X} \leqslant \mathcal{Y}$ *iff* $\mathcal{X} \in \mathrm{Pref}(\mathcal{Y})$
- $\lambda(\mathcal{P}|_{\downarrow e}) = \nu(e)$ *for* $\mathcal{P} = (F, \preceq, \nu) \in \mathfrak{P}, e \in F$
- $\mathcal{X} \# \mathcal{Y}$ *iff there is no* $\mathcal{P} \in \mathfrak{P}$ *with* $\mathcal{X}, \mathcal{Y} \in \mathrm{Pref}(\mathcal{P})$
- $\mathcal{X} \in \mathrm{Term}$ *iff there is* $\mathcal{P} = (F, \preceq, \nu) \in \mathfrak{P}$ *such that* $\mathcal{X} = \{\mathcal{P}|_{\downarrow e} : e \in F\}$

To show that $\mathcal{E}[\mathfrak{P}]$ is well defined, we note that $(E, \leqslant, \lambda)$ is a (possibly infinite) pomset as the prefix relation on any pomset family is a partial order, and $\#$ is clearly irreflexive and symmetric. Furthermore, the principle of finite causes holds as $\mathcal{P} = (F, \preceq, \nu) \in \mathfrak{P}$ is finite and thus, $\mathrm{Pref}(\mathcal{P}|_{\downarrow e})$ is also finite for all $e \in F$. To show that conflict heredity holds, let $\mathcal{X}, \mathcal{Y}, \mathcal{Z} \in E$ and $\mathcal{X} \# \mathcal{Y}$, $\mathcal{Y} \leqslant \mathcal{Z}$ and assume that $\mathcal{X} \# \mathcal{Z}$ does not hold. Then, there is $\mathcal{P} \in \mathfrak{P}$ such that $\mathcal{X}, \mathcal{Z} \in \mathrm{Pref}(\mathcal{P})$. By the definition of $\leqslant$, $\mathcal{Y} \in \mathrm{Pref}(\mathcal{Z}) \subseteq \mathrm{Pref}(\mathcal{P})$, which contradicts $\mathcal{X} \# \mathcal{Y}$ as there should be no $\mathcal{Q} \in \mathfrak{P}$ with $\mathcal{X}, \mathcal{Y} \in \mathrm{Pref}(\mathcal{Q})$, violated by $\mathcal{Q} = \mathcal{P}$. It is left to show that $\mathrm{Term} \subseteq \mathrm{Conf}\big(\mathcal{E}[\mathfrak{P}]\big)$, which is a direct consequence of the following lemma.

Lemma 3. *For all* $\mathcal{P} = (F, \preceq, \nu) \in \mathrm{Pref}(\mathfrak{P})$, $C = \big\{\mathcal{P}|_{\downarrow e} : e \in F\big\}$, *we have* $C \in \mathrm{Conf}\big(\mathcal{E}[\mathfrak{P}]\big)$ *and* $\mathcal{P} = (C, \leqslant|_C, \lambda|_C)$.

Proof. Towards an induction on $n = |F|$, the statement is clearly fulfilled for $n = 0$ by $\varnothing \in \mathrm{Conf}\big(\mathcal{E}[\mathfrak{P}]\big)$. Now, let $|F| = n{+}1$ and assume that the statement holds

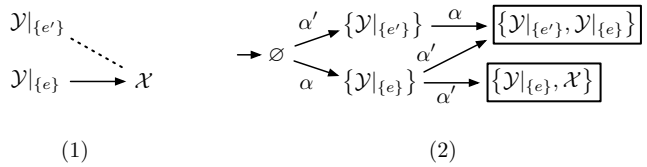
for all $\mathcal{Q} \in \mathrm{Pref}(\mathfrak{P})$ with an event space containing n elements. In particular, for all $\alpha \in \Sigma$ with $\mathcal{P}' = (F', \preceq|_{F'}, \nu|_{F'}) \in \mathrm{Pref}_\alpha(\mathcal{P})$ there is an $f \in F$ with $\nu(f) = \alpha$ such that $F \backslash F' = \{f\}$. Since we assume pomsets to be not autoconcurrent, f is uniquely defined if it exists. We first show that $C = C' \cup \{\mathcal{P}|_{\downarrow f}\} \in \mathrm{Conf}\big(\mathcal{E}[\mathfrak{P}]\big)$ with $C' = \{\mathcal{P}'|_{\downarrow e} : e \in F'\}$. Since $\mathcal{P} \in \mathrm{Pref}(\mathfrak{P})$ we have $\mathcal{P}|_{\downarrow f} \in E$. Furthermore, $C' \in \mathrm{Conf}\big(\mathcal{E}[\mathfrak{P}]\big)$ by induction hypothesis and thus $C \subseteq E$. C is conflict-free with $\mathcal{P}$ as witness. Now assume that C is not downward-closed, i.e., there is an $\mathcal{X} \in E \backslash C$ with $\mathcal{X} \leqslant \mathcal{P}|_{\downarrow f}$. By the definition of $\leqslant$ we have $\mathcal{X} \in \mathrm{Pref}(\mathcal{P}|_{\downarrow f})$. Thus, there is an $x \in F$ with $\mathcal{X} = \mathcal{P}|_{\downarrow x}$. If $x \neq f$, then $\mathcal{X} \in C'$ and if $x = f$, then $\mathcal{X} = \mathcal{P}|_{\downarrow f}$. Hence, $\mathcal{X} \in C$, contradicting $\mathcal{X} \in E \backslash C$. Now we show that $\mathcal{P} = (C, \leqslant|_C, \lambda|_C)$. By induction hypothesis, we have $\mathcal{P}' = (C', \leqslant|_{C'}, \lambda|_{C'})$. Thus, it suffices to show that for all $e \in F$ we have $e \preceq f$ iff $\mathcal{P}|_{\downarrow e} \leqslant \mathcal{P}|_{\downarrow f}$:

($\Rightarrow$) It follows directly that $\downarrow e \subseteq \downarrow f$ and thus, $\mathcal{P}|_{\downarrow e} \in \mathrm{Pref}(\mathcal{P}|_{\downarrow f})$.
($\Leftarrow$) From $\mathcal{P}|_{\downarrow e} \in \mathrm{Pref}(\mathcal{P}|_{\downarrow f})$, we get $\downarrow e \subseteq \downarrow f$ and thus, $e' \preceq f$ for all $e' \in \downarrow e$. □

4.1 Properties of pomset event structures

In the general case, pomset event structures do not induce a deterministic transition system such they do not obey delayed choice in the sense of Definition 4. We illustrate this fact by the following example.

Example 6. Let us reconsider the example of Section 3.3. On the left of Figure 6, the pomset event structure of $\{\mathcal{X}, \mathcal{Y}\}$ is depicted (1), where the arrow connects



(1) (2)

Fig. 6. $\mathcal{E}\big[\{\mathcal{X}, \mathcal{Y}\}\big]$ (1) and induced transition system $\mathfrak{ts}\big(\mathcal{E}[\{\mathcal{X}, \mathcal{Y}\}]\big)$ (2)

causal dependent events and the dashed line conflicting ones. On the right of Figure 6, the induced transition system is shown (2). Note that this transition system is non-deterministic in the configuration $\big\{\mathcal{Y}|_{\{e\}}\big\}$.

We now present a further lemma that intuitively provides the backward direction of Lemma 3:

Lemma 4. *For all* $C \in \mathrm{Conf}\big(\mathcal{E}[\mathfrak{P}]\big)$ *we have* $(C, \leqslant|_C, \lambda|_C) \in \mathrm{Pref}(\mathfrak{P})$.

Proof. Since C is conflict-free there is a $\mathcal{P} = (F, \preceq, \nu) \in \mathrm{Pref}(\mathfrak{P})$ such that $\mathcal{X} \in \mathrm{Pref}(\mathcal{P})$ for all $\mathcal{X} \in C$. Thus, there is a function $\xi \colon C \to F$ such that for all $\mathcal{X} \in C$ we have $\mathcal{X} = \mathcal{P}|_{\downarrow \xi(\mathcal{X})}$. Clearly, ξ is injective and it is left to show that $\mathcal{P}|_{\xi(C)} = (C, \leqslant|_C, \lambda|_C)$. We do so by showing that for all $\mathcal{X}, \mathcal{Y} \in C$ we have $\xi(\mathcal{X}) \preceq \xi(\mathcal{Y})$ iff $\mathcal{X} \leqslant \mathcal{Y}$:

($\Rightarrow$) From $\downarrow\xi(\mathcal{X}) \subseteq \downarrow\xi(\mathcal{Y})$, we get $\mathcal{P}|_{\xi(\mathcal{X})} \in \mathrm{Pref}(\mathcal{P}|_{\xi(\mathcal{Y})})$ and hence, $\mathcal{X} \in \mathrm{Pref}(\mathcal{Y})$.
($\Leftarrow$) As $\mathcal{X} \in \mathrm{Pref}(\mathcal{Y})$, we have $\mathcal{P}|_{\xi(\mathcal{X})} \in \mathrm{Pref}(\mathcal{P}|_{\xi(\mathcal{Y})})$ and thus, $\downarrow\xi(\mathcal{X}) \subseteq \downarrow\xi(\mathcal{Y})$. Hence, for all $e' \in \downarrow\xi(\mathcal{X})$ we get $e' \preceq \xi(\mathcal{Y})$ and in particular $\xi(\mathcal{X}) \preceq \xi(\mathcal{Y})$. □

Mainly relying on Lemma 3 and the Lemma 4 above, we show compatibility of $\mathcal{E}[\mathfrak{P}]$ with its generating pomset $\mathfrak{P}$:

Theorem 2 (Compatibility Theorem). $\mathfrak{pof}\big(\mathcal{E}[\mathfrak{P}]\big) = \mathfrak{P}$.

Proof. ($\subseteq$) For all $\mathcal{P} \in \mathfrak{pof}\big(\mathcal{E}[\mathfrak{P}]\big)$ there is some $C \in \mathrm{Term}$ with $\mathcal{P} = (C, \leqslant|_C, \lambda|_C)$. By Lemma 4 we have $\mathcal{P} \in \mathrm{Pref}(\mathfrak{P})$ and due to the definition of Term in $\mathcal{E}[\mathfrak{P}]$, we finally obtain $\mathcal{P} \in \mathfrak{P}$.
($\supseteq$) Let $\mathcal{P} = (F, \leq, \nu) \in \mathfrak{P}$ and $C = \{\mathcal{P}|_{\downarrow e} : e \in F\}$. Then, due to Lemma 3, $\mathcal{P} = (C, \leqslant|_C, \lambda|_C)$ and $C \in \mathrm{Term}$. Thus, by the definition of $\mathfrak{pof}$, we get $\mathcal{P} \in \mathfrak{pof}\big(\mathcal{E}[\mathfrak{P}]\big)$. □

4.2 Pomset event structures for MSGs

As any MSG $\mathcal{G}$ induces a pomset semantics $\mathfrak{P}[\mathcal{G}]$, an event structure semantics for $\mathcal{G}$ is naturally defined through $\mathcal{E}\big[\mathfrak{P}[\mathcal{G}]\big]$.

Example 7. Let us consider the running example with the MSG $\mathcal{G}_{\mathrm{bsp}}$ from Example 3 and denote its event structure by $\mathcal{E}\big[\mathfrak{P}[\mathcal{G}_{\mathrm{bsp}}]\big] = (E, \leqslant, \lambda, \#, \mathrm{Term})$. Figure 7 shows a fragment of $\mathcal{E}\big[\mathfrak{P}[\mathcal{G}_{\mathrm{bsp}}]\big]$. Arrows indicate direct successors, i.e., $e \to e'$ iff $e < e'$ and there is no $e'' \in E$ with $e < e'' < e'$. Dashed lines connect minimal conflicting events, i.e., $e \dashrightarrow e'$ iff $e \# e'$ and there is no $e'' \in E$ with $e \# e'' < e'$ or $e' \# e'' < e$. All other conflicting events can be derived from these minimal conflicting events through conflict heredity. Note that, e.g., the event $\mathcal{M}_a$ has

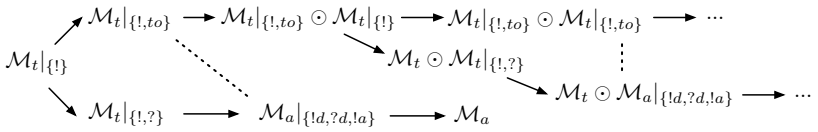


Fig. 7. Fragment of the event structure for $\mathfrak{P}[\mathcal{G}_{\mathrm{bsp}}]$

no successor and is in conflict with every event of the upper branch of Figure 7. Thus, the configuration $C = \{\mathcal{M}_t|_{\{!\}}, \mathcal{M}_t|_{\{!,?\}}, \mathcal{M}_a|_{\{!d,?d,!a\}}, \mathcal{M}_a\}$ is maximal in the sense that it cannot be extended by any other event. Furthermore, $C \in \mathrm{Term}$ as $\mathcal{M}_a \in \mathfrak{P}$.

Note that the basis for our construction in Definition 6 is provided by pomset downward closures, which in the setting of MSGs correspond to p-views for processes $p \in P$ [15]. Although the transition system induced by a pomset event structure does neither need to be deterministic nor bisimilar to the corresponding suffix transition system (see Example 6), it obeys delayed choice in the setting of MSGs:

Theorem 3. *Let $\mathcal{G}$ be an MSG. Then, $\mathfrak{ts}\big(\mathcal{E}[\mathfrak{P}[\mathcal{G}]]\big)$ is isomorphic to $\mathcal{T}_{\text{pref}}\big[\mathfrak{P}[\mathcal{G}]\big]$.*

Proof. Let us denote $\mathcal{E}\big[\mathfrak{P}[\mathcal{G}]\big]$ by $\mathcal{E} = (E, \leqslant, \lambda, \#, \text{Term})$ and the transition relation of $\mathfrak{ts}(\mathcal{E})$ by $\longrightarrow$. Furthermore, let $\mathcal{T}_{\text{pref}}\big[\mathfrak{P}[\mathcal{G}]\big] = \big(S, \{\varnothing\}, \rightsquigarrow, \mathfrak{T}\big)$. Lemma 4 induces a mapping $\xi\colon \text{Conf}(\mathcal{E}) \to \text{Pref}(\mathfrak{P}[\mathcal{G}])$ by $\xi(C) = (C, \leqslant|_C, \lambda|_C)$ for all $C \in \text{Conf}(\mathcal{E})$. Due to Lemma 3, ξ is bijective. Since every reachable state $\mathfrak{X}$ in $\mathcal{T}_{\text{pref}}\big[\mathfrak{P}[\mathcal{G}]\big]$ is a singleton (see Lemma 2), it suffices to show that for all $\alpha \in \mathit{Act}$ and $C, D \in \text{Conf}(\mathcal{E})$ we have $C \xrightarrow{\alpha} D$ iff $\big\{\xi(C)\big\} \overset{\alpha}{\rightsquigarrow} \big\{\xi(D)\big\}$.

- $(\Rightarrow)$ For $C \xrightarrow{\alpha} D$ there is an event $e \in E$ such that $D = C \cup \{e\}$ and $\lambda(e) = \alpha$. By the definition of prefixes and the fact that we only consider pomsets that are not autoconcurrent, we thus obtain $\big\{(C, \leqslant|_C, \lambda|_C)\big\} = \text{Pref}_\alpha\big(\{(D, \leqslant|_D, \lambda|_D)\}\big)$. Hence, $\big\{\xi(C)\big\} \overset{\alpha}{\rightsquigarrow} \big\{\xi(D)\big\}$.
- $(\Leftarrow)$ Let $\big\{\xi(C)\big\} \overset{\alpha}{\rightsquigarrow} \big\{\xi(D)\big\}$. Then $\big\{(C, \leqslant|_C, \lambda|_C)\big\} = \text{Pref}_\alpha\big(\{(D, \leqslant|_D, \lambda|_D)\}\big)$ and thus, there is an event $e \in D$ with $\lambda(e) = \alpha$ such that $(C, \leqslant|_C, \lambda|_C) = \big(D\backslash\{e\}, \leqslant|_{D\backslash\{e\}}, \lambda|_{D\backslash\{e\}}\big)$. By Lemma 3 we obtain $C = D\backslash\{e\}$ and hence, the definition of $\mathfrak{ts}(\cdot)$ yields $C \xrightarrow{\alpha} D$.

It is left to show that $C \in \text{Term}$ iff $\big\{\xi(C)\big\} \in \mathfrak{T}$. Due to Theorem 2, we have $C \in \text{Term}$ iff $\xi(C) \in \mathfrak{P}[\mathcal{G}]$. By the definition of $\mathcal{T}_{\text{pref}}\big[\mathfrak{P}[\mathcal{G}]\big]$ and Lemma 2, $\{\mathcal{X}\} \in \mathfrak{T}$ iff $\mathcal{X} \in \mathfrak{P}[\mathcal{G}]$. The statement follows directly since ξ is a bijection. □

As a direct consequence of the above theorem and Theorem 1, we obtain that $\mathcal{E}\big[\mathfrak{P}[\mathcal{G}]\big]$ can be seen as a delayed-choice semantics for $\mathcal{G}$. Thus, our definition of pomset event structures covers the first non-interleaving branching-time semantics for MSGs that follows the delayed-choice principle.

Corollary 1. $\mathfrak{ts}\big(\mathcal{E}[\mathfrak{P}]\big)$ *obeys delayed choice, i.e.,* $\mathfrak{ts}\big(\mathcal{E}[\mathfrak{P}]\big)$ *is deterministic and bisimilar to* $\mathcal{T}_{\text{suff}}[\mathfrak{P}]$.

5 Conclusion

The main contribution of this paper is that we provided a semantical framework of branching-time semantics for pomset families and MSGs following the delayed-choice principle. In contrast to the original definition of delayed choice based on process algebras, we circumvented the intrinsic linear-time operators in terms of delayed choice [3] and weak sequential composition [35] by operating directly on pomset families. Within this approach, delayed choice corresponds to the standard union operation on pomset families that arise from removing minimal events of pomset family members, and weak sequential composition corresponds to local concatenation of pomsets [32] (as intended by [35]). We thus avoid difficulties within the definition of the standard operational semantics for MSG [19,33] that require fixed points over operator-defining rules with negative premises. As a reference semantics, we defined suffix transition systems, which closely follow the process-algebraic approach in the sense that states represent future behaviors. The prefix transition-system semantics provides a connection

to the branching-time semantics defined in [10], where quantitative aspects for MSGs have been investigated. Whereas previously presented event-structure semantics for MSGs [14] do not follow the delayed-choice principle, we constructed an event structure that is consistent with our transition-system semantics, i.e., those transition system is deterministic and bisimilar to our reference semantics.

We illustrated that our event structure semantics follows the delayed-choice principle by referring to its induced transition system. It naturally arises the question whether there is a reasonable definition of delayed choice directly on event structures, possibly relying on deterministic event structures [34]. This question and the problem of defining an event structure obeying delayed choice for arbitrary pomset families is left for further work. Towards an application of our semantical framework, extending LOTOS [7] with a delayed-choice operator could enable reasoning about delayed-choice semantics for pomsets and MSGs.

Acknowledgements. The authors thank Arend Rensink and Joost-Pieter Katoen for their valuable comments on this paper.

References

1. R. Alur, G. J. Holzmann, and D. Peled. An analyzer for message sequence charts. In *Software Concepts and Tools*, pages 304–313, 1996.
2. R. Alur and M. Yannakakis. Model checking of message sequence charts. In *Proc. of CONCUR'99*, volume 1664 of *LNCS*, pages 114–129, 1999.
3. J. C. M. Baeten and S. Mauw. Delayed choice: an operator for joining message sequence charts. In *Proc. of FORTE'94*, pages 340–354, 1994.
4. J. A. Bergstra, I. Bethke, and A. Ponse. Process algebra with iteration and nesting. *The Computer Journal*, 37(4):243, 1994.
5. B. Bollig, D. Kuske, and I. Meinecke. Propositional dynamic logic for message-passing systems. *Logical Methods in Computer Science*, 6(3), 2010.
6. T. Bolognesi and E. Brinksma. Introduction to the ISO specification language LOTOS. *Computer Networks*, 14:25–59, 1987.
7. J. Chakraborty, D. D'Souza, and K. Narayan Kumar. *Analysing Message Sequence Graph Specifications*, volume 6415 of *LNCS*, pages 549–563. Springer, 2010.
8. C. Dubslaff and C. Baier. Quantitative analysis of communication scenarios. In *Proc. of the 13th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS)*, volume 9268 of *Lecture Notes in Computer Science*, pages 76–92. Springer, 2015.
9. B. Genest and A. Muscholl. Message sequence charts: A survey. In *ACSD*, pages 2–4, 2005.
10. J. F. Groote. Transition system specifications with negative premises. *Theor. Comput. Sci.*, 118(2):263–299, 1993.
11. O. M. Group. Unified modeling language (uml): Superstructure version 2.4.1. http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF/, August 2011.
12. L. Hélouët, C. Jard, and B. Caillaud. An event structure based semantics for high-level message sequence charts. *Math. Struct. in Comput. Sci.*, 12(4):377–402, 2002.
13. J. G. Henriksen, M. Mukund, K. N. Kumar, M. Sohoni, and P. S. Thiagarajan. A theory of regular msc languages. *Inf. Comput.*, 202:1–38, October 2005.

14. S. Heymer. A semantics for MSC based on Petri-net components. In *In Proc. 2st Workshop of the SDL Forum Society on SDL and MSC - SAM'2000*, 2000.
15. ITU-T. Message Sequence Chart (MSC). *Z.120 v1.0*, 1993.
16. ITU-T. Message Sequence Chart (MSC). *Z.120 v2.0*, 1996.
17. ITU-T. Annex B: Formal semantics of Message Sequence Charts. *Z.120 v2.2*, 1998.
18. ITU-T. Message Sequence Chart (MSC). *Z.120 v5.0*, 2011.
19. J. Katoen and L. Lambert. Pomsets for message sequence charts. In *8. GI/ITG-Fachgespraech*, pages 197–207. Shaker Verlag, 1998.
20. V. Levin and D. Peled. Verification of message sequence charts via template matching. In *Proc. of TAPSOFT '97*, volume 1214 of *LNCS*, pages 652–666. Springer Verlag, 1997.
21. P. Madhusudan. Reasoning about sequential and branching behaviours of message sequence graphs. In *Proc. of the 28th International Colloquium on Automata, Languages and Programming, ICALP'01*, volume 2076 of *LNCS*, pages 809–820. Springer, 2001.
22. S. Mauw and M. Reniers. An algebraic semantics of basic message sequence charts. *The Computer Journal*, 37:269–277, 1994.
23. S. Mauw and M. Reniers. Operational semantics for msc'96. *Computer Networks*, 31(17):1785 – 1799, 1999.
24. S. Mauw and M. A. Reniers. High-level message sequence charts. In *SDL Forum*, pages 291–306, 1997.
25. R. Milner. *Communication and concurrency*. PHI Series in computer science. Prentice Hall, 1989.
26. A. Muscholl and D. Peled. Message sequence graphs and decision problems on Mazurkiewicz traces. In *Proc. of MFCS'99, LNCS 1672*, pages 81–91. Springer, 1999.
27. A. Muscholl and D. Peled. Deciding properties of message sequence charts. In *Scenarios: Models, Transformations and Tools*, LNCS (3466). Springer Verlag, 2005.
28. A. Muscholl, D. Peled, and Z. Su. Deciding properties for message sequence charts. In *Proceedings of FoSSaCS'98*, number 1378 in LNCS, pages 226–242, 1998.
29. M. Nielsen, G. Plotkin, and G. Winskel. Petri nets, event structures and domains. *Theoretical Computer Science*, 13(1):85 – 108, 1981.
30. V. Pratt. Modeling concurrency with partial orders. *International Journal of Parallel Programming*, 15:33–71, 1986.
31. M. A. Reniers. *Message Sequence Chart: Syntax and Semantics*. PhD thesis, Eindhoven University of Technology, June 1999.
32. A. Rensink. A complete theory of deterministic event structures. In *CONCUR'95: Proc. of the 6th International Conference on Concurrency Theory, Philadelphia, PA, USA, 1995*, pages 160–174, 1995.
33. A. Rensink and H. Wehrheim. *Weak sequential composition in process algebras*, pages 226–241. Springer, Berlin, Heidelberg, 1994.
34. G. Winskel and M. Nielsen. Models for concurrency. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of logic in computer science (vol. 4)*, pages 1–148. Oxford University Press, Oxford, UK, 1995.