Global types and event structure semantics for asynchronous multiparty sessions

Ilaria Castellani, Mariangiola Dezani-Ciancaglini, Paola Giannini

GLOBAL TYPES AND EVENT STRUCTURE SEMANTICS FOR ASYNCHRONOUS MULTIPARTY SESSIONS

ILARIA CASTELLANI, MARIANGIOLA DEZANI-CIANCAGLINI, AND PAOLA GIANNINI

INRIA, Université Côte d'Azur, Sophia Antipolis, France
e-mail address: ilaria.castellani@inria.fr

Dipartimento di Informatica, Università di Torino, Italy
e-mail address: dezani@di.unito.it

Dipartimento di Scienze e Innovazione Tecnologica, Università del Piemonte Orientale, Italy
e-mail address: paola.giannini@uniupo.it

Abstract. We propose an interpretation of multiparty sessions with asynchronous communication as *Flow Event Structures*. We introduce a new notion of global type for asynchronous multiparty sessions, ensuring the expected properties for sessions, including progress. Our global types, which reflect asynchrony more directly than standard global types and are more permissive, are themselves interpreted as *Prime Event Structures*. The main result is that the Event Structure interpretation of a session is equivalent, when the session is typable, to the Event Structure interpretation of its global type.

1. Introduction

Session types describe interactions among a number of participants, which proceed according to a given protocol. They extend classical data types by specifying, in addition to the type of exchanged data, also the interactive behaviour of participants, namely the sequence of their input/output actions towards other participants. The aim of session types is to ensure safety properties for sessions, such as the *absence of communication errors* (no type mismatch in exchanged data) and *deadlock-freedom* (no standstill until every participant is terminated). Sometimes, a stronger property is targeted, called *progress* (no participant waits forever).

Initially conceived for describing binary protocols in the π-calculus [THK94, HVK98], session types have been later extended to multiparty protocols [HYC08, HYC16] and

Key words and phrases: Communication-centric Systems, Process Calculi, Event Structures, Multiparty Session Types.

This research has been supported by the ANR17-CE25-0014-01 CISC project.

Partially supported by EU H2020-644235 Rephrase project, EU H2020-644298 HyVar project, IC1402 ARVI and Ateneo/CSP project RunVar.

This original research has the financial support of the Università del Piemonte Orientale.

Preprint submitted to
Logical Methods in Computer Science

embedded into a range of functional, concurrent, and object-oriented programming languages [ABB+16]. While binary sessions can be described by a single session type, multiparty sessions require two kinds of types: a *global type* that describes the whole session protocol, and *local types* that describe the contributions of the individual participants to the protocol. The key requirement in order to achieve the expected safety properties is that all local types be obtained as projections from the same global type.

Communication in sessions is always directed from a given sender to a given receiver. It can be synchronous or asynchronous. In the first case, sender and receiver need to synchronise in order to exchange a message. In the second case, messages may be sent at any time, hence a sender is never blocked. The sent messages are stored in a queue, where they may be fetched by the intended receiver. Asynchronous communication is often favoured for multiparty sessions, since such sessions may be used to model web services or distributed applications, where the participants are spread over different sites.

Session types have been shown to bear a strong connection with models of concurrency such as communicating automata [DY12], as well as with message-sequence charts [HYC16], graphical choreographies [LTY15, TG18], and various brands of linear logics [CP10, TCP11, Wad14, PCPT14, CPT16].

In a companion paper [CDG19a], we investigated the relationship between synchronous multiparty sessions and Event Structures (ESs) [Win88], a well-known model of concurrency which is grounded on the notions of causality and conflict between events. We considered a simple calculus, where sessions are described as networks of sequential processes [DCGJ+16], equipped with standard global types [HYC08]. We proposed an interpretation of sessions as *Flow Event Structures* (FESs) [BC88, BC94], as well as an interpretation of global types as *Prime Event Structures* (PESs) [Win80, NPW81]. We showed that for typed sessions these two interpretations agree, in the sense that they yield isomorphic domains of configurations.

In the present paper, we undertake a similar endeavour in the asynchronous setting. This involves devising a new notion of global type for asynchronous networks. We start by considering a core session calculus as in the synchronous case, where processes are only able to exchange labels, not values, hence local types coincide with processes and global types may be directly projected to processes. Moreover, networks are now endowed with a queue and they act on this queue by performing outputs or inputs: an output stores a message in the queue, while an input fetches a message from the queue.

To illustrate the difference between synchronous and asynchronous sessions and motivate the introduction of new global types for the latter, let us discuss a simple example. Consider the network:

$$\mathsf{N} = \mathsf{p}[\![\, \mathsf{q}!\lambda; \mathsf{q}?\lambda' \,]\!] \parallel \mathsf{q}[\![\, \mathsf{p}!\lambda'; \mathsf{p}?\lambda \,]\!]$$

where each of the participants p and q wishes to first send a message to the other one and then receive a message from the other one.

In a synchronous setting this network is stuck, because a network communication arises from the synchronisation of an output with a matching input, and here the output $\mathsf{q}!\lambda$ of p cannot synchronise with the input $\mathsf{p}?\lambda$ of q, since the latter is guarded by the output $\mathsf{p}!\lambda'$. Similarly, the output $\mathsf{p}!\lambda'$ of q cannot synchronise with the input $\mathsf{q}?\lambda'$ of p. Indeed, this network is not typable because any global type for it should have one of the two forms:

$$\mathsf{G}_1 = \mathsf{p} \to \mathsf{q} : \lambda;\ \mathsf{q} \to \mathsf{p} : \lambda' \qquad\qquad \mathsf{G}_2 = \mathsf{q} \to \mathsf{p} : \lambda';\ \mathsf{p} \to \mathsf{q} : \lambda$$

However, neither of the G_i projects down to the correct processes for both p and q in N. For instance, G_1 projects to the correct process $\mathsf{q}!\lambda \cdot \mathsf{q}?\lambda'$ for p, but its projection on q is $\mathsf{p}?\lambda \cdot \mathsf{p}!\lambda'$, which is not the correct process for q.

In an asynchronous setting, on the other hand, this network is run in parallel with a queue $\mathcal{M}$, which we indicate by $\mathsf{N} \parallel \mathcal{M}$, and it can always move for whatever choice of $\mathcal{M}$. Indeed, the moves of an asynchronous network are no more complete communications but rather "communication halves", namely outputs or inputs. For instance, if the queue is empty, then $\mathsf{N} \parallel \emptyset$ can move by first performing the two outputs in any order, and then the two inputs in any order. If instead the queue contains a message from p to q with label λ_1, followed by a message from q to p with label λ_2, which we indicate by $\mathcal{M} = \langle \mathsf{p}, \lambda_1, \mathsf{q} \rangle \cdot \langle \mathsf{q}, \lambda_2, \mathsf{p} \rangle$, then the network will be stuck after performing the two outputs, since the two messages on top of the queue will not be those expected by p and q. Hence we look for a notion of global type that accepts the network $\mathsf{N} \parallel \emptyset$ but rejects the network $\mathsf{N} \parallel \langle \mathsf{p}, \lambda_1, \mathsf{q} \rangle \cdot \langle \mathsf{q}, \lambda_2, \mathsf{p} \rangle$.

The idea for our new *asynchronous global types* is quite simple: to split communications into outputs and inputs, and to add a queue to the type, thus mimicking very closely the behaviour of asynchronous networks. Hence, our global types have the form $\mathsf{G} \parallel \mathcal{M}$. Clearly, we must impose some well formedness conditions on such global types, taking into account also the content of the queue. Essentially, this amounts to requiring that each input appearing in the type be justified by a preceding output or by a message in the queue, and vice versa, that each output or message in the queue be matched by a corresponding input.

With our new global types, it becomes now possible to type the network $\mathsf{N} \parallel \emptyset$ with the asynchronous global type $\mathsf{G} \parallel \emptyset$, where $\mathsf{G} = \mathsf{pq}!\lambda; \mathsf{qp}!\lambda'; \mathsf{pq}?\lambda; \mathsf{qp}?\lambda'$, or with the other global types obtained from it by swapping the outputs or the inputs. Instead, the network $\mathsf{N} \parallel \langle \mathsf{p}, \lambda_1, \mathsf{q} \rangle \cdot \langle \mathsf{q}, \lambda_2, \mathsf{p} \rangle$ will be rejected because the global type $\mathsf{G} \parallel \langle \mathsf{p}, \lambda_1, \mathsf{q} \rangle \cdot \langle \mathsf{q}, \lambda_2, \mathsf{p} \rangle$ is not well formed, since its two inputs do not match the first two messages in the queue.

A different solution was proposed in [MYH09] by means of an asynchronous subtyping relation on local types which allows outputs to be anticipated. In our setting this boils down to a subtyping relation on processes yielding both $\mathsf{q}!\lambda; \mathsf{q}?\lambda' \leq \mathsf{q}?\lambda'; \mathsf{q}!\lambda$ and $\mathsf{p}!\lambda'; \mathsf{p}?\lambda \leq \mathsf{p}?\lambda; \mathsf{p}!\lambda'$. With the help of this subtyping, both G_1 and G_2 become types for the network $\mathsf{N} \parallel \emptyset$ above. Unfortunately, however, this subtyping turned out to be undecidable [BCZ17, LY17].

To define our interpretations of asynchronous networks and asynchronous global types into Flow and Prime Event Structures, respectively, we follow the same schema as for their synchronous counterparts in our previous work [CDG19a]. In particular, the events of the ESs are defined syntactically and they record their "history". More specifically, the events of the FES associated with a network record their local history (namely the past actions of the involved participants), while the events of the PES associated with a global type record the global history of the computation (the whole sequence of past communications) and thus they must be quotiented by a permutation equivalence. However, while in [CDG19a] an event represented a communication between two participants, here it represents an output or an input pertaining to a single participant. As a consequence, some care must be taken in defining the causality relation, and in particular the "cross-causality" between an output and the matching input, which are events pertaining to different participants. A further complication is due to the presence of the queue, which appears inside the events themselves. Therefore, our ES semantics for the asynchronous setting is far from

being a trivial adaptation of that given in [CDG19a] for the synchronous setting. To our knowledge, this is also the first time Event Structures are used to give semantics to an asynchronous calculus. We should mention, however, that a semantics for finite asynchronous choreographies by means of PESs was recently proposed in [LMT20].

To sum up, the contribution of this paper is twofold:

1) We propose an original syntax for asynchronous global types, which, in our view, models asynchronous communication in a more natural way than existing approaches. In the literature, typing rules for asynchronous multiparty session types use the standard syntax of global types, as introduced in [HYC16]. Typability of asynchronous networks is enhanced by the subtyping first proposed in [MYH09], which, however, was later shown to be undecidable [BCZ17, LY17]. Our type system is more permissive than the standard one [HYC16] – in particular, since it allows outputs to take precedence over inputs as in [MYH09], a characteristics of asynchronous communication – but it remains decidable. We show that our new global types ensure classical safety properties as well as progress.

2) We present an Event Structure semantics for asynchronous networks and for our new asynchronous global types. Networks are interpreted as FESs and asynchronous global types are interpreted as PESs. Our main result here is an isomorphism between the configuration domains of the FES of a typed network and the PES of its global type.

The paper is organised as follows. Section 2 introduces our calculus for asynchronous multiparty sessions. In Section 3 we recap from previous work the necessary material about Event Structures. In Section 4 we recall our interpretation of processes as PESs, taken from our companion paper [CDG19a]. In Section 5 we present our interpretation of asynchronous networks as FESs. Section 6 introduces our new global types and the associated type system, establishing its main properties. In Section 7 we define our interpretation of asynchronous global types as PESs. Finally, in Section 8 we prove the equivalence between the FES semantics of a network and the PES semantics of its global type. We conclude with a discussion on related work and future research directions in Section 9.

2. A Core Calculus for Multiparty Sessions

We now formally introduce our calculus, where multiparty sessions are represented as networks of processes. The operational semantics is based on *asynchronous communication*, where message emission is non-blocking and sent messages are stored in a queue while waiting to be read by their receiver.

We assume the following base sets: *participants*, ranged over by $\mathsf{p}, \mathsf{q}, \mathsf{r}$ and forming the set Part, and *labels*, ranged over by $\lambda, \lambda', \ldots$ and forming the set Lab.

Let $\pi \in \{\mathsf{p}!\lambda, \mathsf{p}?\lambda \mid \mathsf{p} \in \mathsf{Part}, \lambda \in \mathsf{Lab}\}$ denote an *atomic action*. The action $\mathsf{p}!\lambda$ represents an output of label λ to participant p, while the action $\mathsf{p}?\lambda$ represents an input of label λ from participant p.

Definition 2.1 (Processes and their tree representations)**.** *Processes are defined in three steps:*

(1) Pre-processes *are coinductively defined by:*

$$P \quad ::=^{coind} \quad \bigoplus_{i\in I} \mathsf{p}!\lambda_i; P_i \;\mid\; \sum_{i\in I} \mathsf{p}?\lambda_i; P_i \;\mid\; \mathbf{0}$$

where I is non-empty and $\lambda_j \neq \lambda_h$ for all $j, h \in I$, $j \neq h$, i.e. labels in choices are all different.

(2) *The* tree representation *of a pre-process is a directed rooted tree, where: (a) each internal node is decorated by* $\mathsf{p}!$ *or* $\mathsf{p}?$ *and has as many children as the number of branches of the choice, (b)*

the edge from $\mathsf{p}!$ *or* $\mathsf{p}?$ *to the child* P_i *is decorated by* λ_i *and (c) the leaves of the tree (if any) are decorated by* $\mathbf{0}$.

(3) *A* process *is a pre-process whose tree representation is regular (namely, it has finitely many distinct sub-trees). Processes of the shape* $\bigoplus_{i\in I} \mathsf{p}!\lambda_i; P_i$ *and* $\sum_{i\in I}\mathsf{p}?\lambda_i; P_i$ *are called* output *and* input processes, *respectively.*

In the following, we will omit trailing $\mathbf{0}$'s when writing processes. We identify processes with their tree representations and we shall sometimes refer to the trees as the processes themselves. The regularity condition implies that we only consider processes admitting a finite description. Internal choice ($\bigoplus$) and external choice ($\sum$) are assumed to be associative and commutative, so the order of children of a node does not matter.

Observe that, because of the condition $\lambda_j \neq \lambda_h$ in choices, each path starting from the root in the tree representation of a process is uniquely identified by the sequence of decorations along its nodes and edges.

Note that identifying processes with their tree representations is equivalent to writing processes with the μ-notation and considering them up to an equality which allows for an infinite number of unfoldings. This is also called the equirecursive approach, since it views processes as the unique solutions of recursive equations [Pie02] (Section 20.2). The existence and uniqueness of a solution follow from known results (see [Cou83] and also Theorem 7.5.34 of [CC13]). When writing processes, we shall use (mutually) recursive equations.

In a full-fledged calculus, labels would carry values, namely the exchanges between participants would be of the form $\lambda(v)$. For simplicity, we consider only labels here. This will allow us to project global types directly to processes, without having to explicitly introduce local types, see Section 6.

In our calculus, sent labels are stored in a queue together with sender and receiver names, from where they are subsequently fetched by the receiver[1].

We define *messages* to be triples $\langle \mathsf{p}, \lambda, \mathsf{q}\rangle$ and *message queues* (or simply *queues*) to be possibly empty sequences of messages:

$$\mathcal{M} ::= \emptyset \mid \langle \mathsf{p}, \lambda, \mathsf{q}\rangle \cdot \mathcal{M}$$

The order of messages in the queue is the order in which they will be read. Since the only reading order that matters is that between messages with the same sender and the same receiver, we consider message queues modulo the structural equivalence given by:

$$\mathcal{M} \cdot \langle \mathsf{p}, \lambda, \mathsf{q}\rangle \cdot \langle \mathsf{r}, \lambda', \mathsf{s}\rangle \cdot \mathcal{M}' \equiv \mathcal{M} \cdot \langle \mathsf{r}, \lambda', \mathsf{s}\rangle \cdot \langle \mathsf{p}, \lambda, \mathsf{q}\rangle \cdot \mathcal{M}' \text{ if } \mathsf{p} \neq \mathsf{r} \text{ or } \mathsf{q} \neq \mathsf{s}$$

The equivalence $\equiv$ says that a global queue may be split into a set of *reading queues*, one for each participant (in which messages with different senders are not ordered), or even further, into a set of channel queues, one for each unidirectional channel pq or ordered pair (p, q) of participants.

Note in particular that $\langle \mathsf{p}, \lambda, \mathsf{q}\rangle \cdot \langle \mathsf{q}, \lambda', \mathsf{p}\rangle \equiv \langle \mathsf{q}, \lambda', \mathsf{p}\rangle \cdot \langle \mathsf{p}, \lambda, \mathsf{q}\rangle$. These two equivalent queues represent a situation in which both participants p and q have sent a label to the other one, and neither of them has read the label sent by the other one. This situation may indeed happen in a network with asynchronous communication. Since the two sends occur in parallel, namely, in any order, the order of the corresponding messages in the queue should be irrelevant. This point will be further illustrated by Example 2.4.

[1] We need a queue instead of a multiset, because we want labels between two participants to be read in the same order in which they are sent.

$$\mathsf{p}[\![\bigoplus_{i\in I} \mathsf{q}!\lambda_i; P_i]\!] \parallel \mathsf{N} \parallel \mathcal{M} \xrightarrow{\mathsf{pq}!\lambda_k} \mathsf{p}[\![P_k]\!] \parallel \mathsf{N} \parallel \mathcal{M} \cdot \langle \mathsf{p}, \lambda_k, \mathsf{q} \rangle \quad \text{where } k \in I \quad [\textsc{Send}]$$

$$\mathsf{q}[\![\Sigma_{j\in J} \mathsf{p}?\lambda_j; Q_j]\!] \parallel \mathsf{N} \parallel \langle \mathsf{p}, \lambda_k, \mathsf{q} \rangle \cdot \mathcal{M} \xrightarrow{\mathsf{pq}?\lambda_k} \mathsf{q}[\![Q_k]\!] \parallel \mathsf{N} \parallel \mathcal{M} \quad \text{where } k \in J \quad [\textsc{Rcv}]$$

Figure 1: LTS for networks.

Networks are comprised of located processes of the form $\mathsf{p}[\![P]\!]$ composed in parallel, each with a different participant p, and by a message queue.

Definition 2.2 (Networks)**.** Networks *are defined by:*

$$\mathsf{N} \parallel \mathcal{M}$$

where $\mathsf{N} = \mathsf{p}_1[\![P_1]\!] \parallel \cdots \parallel \mathsf{p}_n[\![P_n]\!]$ *with* $\mathsf{p}_i \neq \mathsf{p}_j$ *for any* $i \neq j$*, and* $\mathcal{M}$ *is a message queue.*

We assume the standard structural congruence on networks, stating that parallel composition is associative and commutative and has neutral element $\mathsf{p}[\![\mathbf{0}]\!]$ for any fresh p.

If $P \neq \mathbf{0}$ we write $\mathsf{p}[\![P]\!] \in \mathsf{N}$ as short for $\mathsf{N} \equiv \mathsf{p}[\![P]\!] \parallel \mathsf{N}'$ for some N'. This abbreviation is justified by the associativity and commutativity of $\parallel$.

To define the *operational semantics* of networks, we use an LTS whose transitions are decorated by inputs or outputs. Therefore, we define the set of *input/output communications* (communications for short), ranged over by β, β', to be $\{\mathsf{pq}!\lambda, \mathsf{pq}?\lambda \mid \mathsf{p}, \mathsf{q} \in \mathsf{Part}, \lambda \in \mathsf{Lab}\}$, where $\mathsf{pq}!\lambda$ represents the emission of a label λ from participant p to participant q, and $\mathsf{pq}?\lambda$ the actual reading by participant q of the label λ sent by participant p. To memorise this notation, it is helpful to view pq as the channel from p to q and the exclamation/question mark as the mode (write/read) in which the channel is used. The LTS semantics of networks is specified by the two Rules [Send] and [Rcv] given in Figure 1. Rule [Send] allows a participant p with an internal choice (a sender) to send to a participant q one of its possible labels λ_i by adding it to the queue. Symmetrically, Rule [Rcv] allows a participant q with an external choice (a receiver) to read the first label λ_k sent to her by participant p, provided this label is among the λ_j's specified in the choice. Thanks to structural equivalence, this message can always be moved to the top of the queue.

A key role in this paper is played by (possibly empty) sequences of communications. As usual we define them as traces.

Definition 2.3 (Traces)**.** *(Finite) traces are defined by:*

$$\tau ::= \epsilon \mid \beta \cdot \tau$$

We use $|\tau|$ *to denote the length of the trace* τ*.*

When $\tau = \beta_1 \cdot \ldots \cdot \beta_n$ $(n \geq 1)$ we write $\mathsf{N} \parallel \mathcal{M} \xrightarrow{\tau} \mathsf{N}' \parallel \mathcal{M}'$ as short for

$$\mathsf{N} \parallel \mathcal{M} \xrightarrow{\beta_1} \mathsf{N}_1 \parallel \mathcal{M}_1 \cdots \mathsf{N}_{n-1} \parallel \mathcal{M}_{n-1} \xrightarrow{\beta_n} \mathsf{N}' \parallel \mathcal{M}'$$

In the following example, we consider the semantics of the network $\mathsf{N} \parallel \emptyset$ discussed in the introduction.

Example 2.4 . *Consider the network:* $\mathsf{N} \parallel \emptyset$*, where* $\mathsf{N} = \mathsf{p}[\![\mathsf{q}!\lambda; \mathsf{q}?\lambda']\!] \parallel \mathsf{q}[\![\mathsf{p}!\lambda'; \mathsf{p}?\lambda]\!]$*. Then* $\mathsf{N} \parallel \emptyset$ *can move by first performing the two sends, in any order, and then the two reads, in any order.*

A possible execution of $\mathsf{N} \parallel \emptyset$ *is:*

$$\begin{array}{rl}
\mathsf{N} \parallel \emptyset & \xrightarrow{\mathsf{pq}!\lambda} \quad \mathsf{p}[\![\, \mathsf{q}?\lambda' \,]\!] \parallel \mathsf{q}[\![\, \mathsf{p}!\lambda'; \mathsf{p}?\lambda \,]\!] \parallel \langle \mathsf{p}, \lambda, \mathsf{q} \rangle \\
& \xrightarrow{\mathsf{qp}!\lambda'} \quad \mathsf{p}[\![\, \mathsf{q}?\lambda' \,]\!] \parallel \mathsf{q}[\![\, \mathsf{p}?\lambda \,]\!] \parallel \langle \mathsf{p}, \lambda, \mathsf{q} \rangle \cdot \langle \mathsf{q}, \lambda', \mathsf{p} \rangle \\
& \equiv \quad \mathsf{p}[\![\, \mathsf{q}?\lambda' \,]\!] \parallel \mathsf{q}[\![\, \mathsf{p}?\lambda \,]\!] \parallel \langle \mathsf{q}, \lambda', \mathsf{p} \rangle \cdot \langle \mathsf{p}, \lambda, \mathsf{q} \rangle \\
& \xrightarrow{\mathsf{qp}?\lambda'} \quad \mathsf{p}[\![\, \mathbf{0} \,]\!] \parallel \mathsf{q}[\![\, \mathsf{p}?\lambda \,]\!] \parallel \langle \mathsf{p}, \lambda, \mathsf{q} \rangle \\
& \xrightarrow{\mathsf{pq}?\lambda} \quad \mathsf{p}[\![\, \mathbf{0} \,]\!] \parallel \mathsf{q}[\![\, \mathbf{0} \,]\!] \parallel \emptyset
\end{array}$$

Note the use of $\equiv$*, allowing label* λ' *to be read by* p *before label* λ *is read by* q*.*

We now introduce the notion of player, which will be extensively used in the rest of the paper. A player of a communication β is a participant who is active in β.

Definition 2.5 (Players of communications)**.** *We denote by* $\mathsf{play}(\beta)$ *the* sets of players of communication β *defined by*

$$\mathsf{play}(\mathsf{pq}!\lambda) = \{\mathsf{p}\} \qquad \mathsf{play}(\mathsf{pq}?\lambda) = \{\mathsf{q}\}$$

The function play is extended to traces in the obvious way:

$$\mathsf{play}(\epsilon) = \emptyset \qquad \mathsf{play}(\beta \cdot \tau) = \mathsf{play}(\beta) \cup \mathsf{play}(\tau)$$

In Section 6 we will use the same notation for the players of a global type. In all cases, the context should make it easy to understand which function is in use.

3. Event Structures

We recall now the definitions of *Prime Event Structure* (PES) from [NPW81] and *Flow Event Structure* (FES) from [BC88]. The class of FESs is more general than that of PESs: for a precise comparison of various classes of event structures, we refer the reader to [BC91]. As we shall see in Sections 4 and 5, while PESs are sufficient to interpret processes, the generality of FESs is needed to interpret networks.

Definition 3.1 (Prime Event Structure)**.** *A prime event structure* (PES) *is a tuple* $S = (E, \leq, \#)$ *where:*

(1) E *is a denumerable set of events;*
(2) $\leq \subseteq (E \times E)$ *is a partial order relation, called the* causality *relation;*
(3) $\# \subseteq (E \times E)$ *is an irreflexive symmetric relation, called the* conflict *relation, satisfying the property:* $\forall e, e', e'' \in E : e \# e' \leq e'' \Rightarrow e \# e''$ *(*conflict hereditariness*).*

We say that two events are *concurrent* if they are neither causally related nor in conflict.

Definition 3.2 (Flow Event Structure)**.** *A flow event structure* (FES) *is a tuple* $S = (E, \prec, \#)$ *where:*

(1) E *is a denumerable set of events;*
(2) $\prec \subseteq (E \times E)$ *is an irreflexive relation, called the* flow *relation;*
(3) $\# \subseteq (E \times E)$ *is a symmetric relation, called the* conflict *relation.*

Note that the flow relation is not required to be transitive, nor acyclic (its reflexive and transitive closure is just a preorder, not necessarily a partial order). Intuitively, the flow relation represents a possible *direct causality* between two events. Observe also that in a FES the conflict relation is not required to be irreflexive nor hereditary; indeed, FESs

may exhibit self-conflicting events, as well as disjunctive causality (an event may have conflicting causes).

Any PES $S = (E, \leq, \#)$ may be regarded as a FES, with $\prec$ given by $<$ (the strict ordering) or by the covering relation of $\leq$.

We now recall the definition of *configuration* for event structures. Intuitively, a configuration is a set of events having occurred at some stage of the computation. Thus, the semantics of an event structure S is given by its poset of configurations ordered by set inclusion, where $\mathcal{X}_1 \subset \mathcal{X}_2$ means that S may evolve from $\mathcal{X}_1$ to $\mathcal{X}_2$.

Definition 3.3 (PES Configuration)**.** *Let $S = (E, \leq, \#)$ be a prime event structure. A configuration of S is a finite subset $\mathcal{X}$ of E such that:*
(1) *$\mathcal{X}$ is downward-closed: $e' \leq e \in \mathcal{X} \;\Rightarrow\; e' \in \mathcal{X}$;*
(2) *$\mathcal{X}$ is conflict-free: $\forall e, e' \in \mathcal{X}, \neg(e \# e')$.*

The definition of configuration for FESs is slightly more elaborated. For a subset $\mathcal{X}$ of E, let $\prec_{\mathcal{X}}$ be the restriction of the flow relation to $\mathcal{X}$ and $\prec^*_{\mathcal{X}}$ be its transitive and reflexive closure.

Definition 3.4 (FES Configuration)**.** *Let $S = (E, \prec, \#)$ be a flow event structure. A configuration of S is a finite subset $\mathcal{X}$ of E such that:*
(1) *$\mathcal{X}$ is downward-closed up to conflicts: $e' \prec e \in \mathcal{X},\ e' \notin \mathcal{X} \;\Rightarrow\; \exists e'' \in \mathcal{X}.\ e' \# e'' \prec e$;*
(2) *$\mathcal{X}$ is conflict-free: $\forall e, e' \in \mathcal{X}, \neg(e \# e')$;*
(3) *$\mathcal{X}$ has no causality cycles: the relation $\prec^*_{\mathcal{X}}$ is a partial order.*

Condition (2) is the same as for prime event structures. Condition (1) is adapted to account for the more general – non-hereditary – conflict relation. It states that any event appears in a configuration with a "complete set of causes". Condition (3) ensures that any event in a configuration is actually reachable at some stage of the computation.

If S is a prime or flow event structure, we denote by $\mathcal{C}(S)$ its set of configurations. Then, the *domain of configurations* of S is defined as follows:

Definition 3.5 (ES Configuration Domain)**.** *Let S be a prime or flow event structure with set of configurations $\mathcal{C}(S)$. The* domain of configurations *of S is the partially ordered set $\mathcal{D}(S) =_{\mathsf{def}} (\mathcal{C}(S), \subseteq)$.*

We recall from [BC91] a useful characterisation for configurations of FESs, which is based on the notion of proving sequence, defined as follows:

Definition 3.6 (Proving Sequences)**.** *Given a flow event structure $S = (E, \prec, \#)$, a* proving sequence *in S is a sequence $e_1; \cdots; e_n$ of distinct non-conflicting events (i.e. $i \neq j \;\Rightarrow\; e_i \neq e_j$ and $\neg(e_i \# e_j)$ for all i, j) satisfying:*

$$\forall i \leq n\, \forall e \in E : \quad e \prec e_i \;\Rightarrow\; \exists k < i\,.\ \text{either } e = e_k \text{ or } e \# e_k \prec e_i$$

Note that any prefix of a proving sequence is itself a proving sequence.

We have the following characterisation of configurations of FESs in terms of proving sequences.

Proposition 3.7 (Representation of configurations as proving sequences [BC91])**.** *Given a flow event structure $S = (E, \prec, \#)$, a subset $\mathcal{X}$ of E is a configuration of S if and only if it can be enumerated as a proving sequence $e_1; \cdots; e_n$.*

Since PESs may be viewed as particular FESs, we may use Definition 3.6 and Proposition 3.7 both for the FESs associated with networks (see Section 5) and for the PESs associated with asynchronous global types (see Section 7). Note that for a PES the condition of Definition 3.6 simplifies to

$$\forall i \leq n \, \forall e \in E : \quad e < e_i \;\; \Rightarrow \;\; \exists k < i \, . \;\; e = e_k$$

To conclude this section, we recall from [CZ97] the definition of *downward surjectivity* (or *downward-onto*, as it was called there), a property that is required for partial functions between two FESs in order to ensure that they preserve configurations. We will make use of this property in Section 5.

Definition 3.8 (Downward surjectivity)**.** *Let $S_i = (E_i, \prec_i, \#_i)$, be a flow event structure, $i = 0, 1$. Let e_i, e_i' range over E_i, $i = 0, 1$. A partial function $f : E_0 \to E_1$ is* downward surjective *if it satisfies the condition:*

$$e_1 \prec_1 f(e_0) \implies \exists e_0' \in E_0 \, . \, e_1 = f(e_0')$$

4. Event Structure Semantics of Processes

In this section, we present an ES semantics for processes, and show that the obtained ESs are PESs. This semantics, which is borrowed from our companion paper [CDG19a], will be the basis for defining the ES semantics for networks in Section 5.

We start by introducing process events, which are non-empty sequences of atomic actions π as defined at the beginning of Section 2.

Definition 4.1 (Process event)**.** Process events (p-events for short) η, η' *are defined by:*

$$\eta \quad ::= \pi \;\mid\; \pi \cdot \eta$$

We denote by $\mathcal{PE}$ the set of p-events.

Let ζ denote a (possibly empty) sequence of actions, and $\sqsubseteq$ denote the prefix ordering on such sequences. Each p-event η may be written either in the form $\eta = \pi \cdot \zeta$ or in the form $\eta = \zeta \cdot \pi$. We shall feel free to use any of these forms. When a p-event is written as $\eta = \zeta \cdot \pi$, then ζ may be viewed as the *causal history* of η, namely the sequence of actions that must have been executed by the process for η to be able to happen.

We define the *action* of a p-event to be its last atomic action:

$$\mathsf{act}(\zeta \cdot \pi) = \pi$$

A p-event η is an *output p-event* if $\mathsf{act}(\eta)$ is an output and an *input p-event* if $\mathsf{act}(\eta)$ is an input.

Definition 4.2 (Causality and conflict relations on p-events)**.** *The* causality *relation $\leq$ and the* conflict *relation $\#$ on the set of p-events $\mathcal{PE}$ are defined by:*

(1) $\eta \sqsubseteq \eta' \;\; \Rightarrow \;\; \eta \leq \eta'$;

(2) $\pi \neq \pi' \Rightarrow \zeta \cdot \pi \cdot \zeta' \; \# \; \zeta \cdot \pi' \cdot \zeta''$.

Definition 4.3 (Event Structure of a process)**.** *The* event structure of process P *is the triple*

$$\mathcal{S}^{\mathcal{P}}(P) = (\mathcal{PE}(P), \leq_P, \#_P)$$

where:

(1) $\mathcal{PE}(P) \subseteq \mathcal{PE}$ *is the set of sequences of decorations along the nodes and edges of a path from the root to an edge in the tree of P;*

(2) $\leq_P$ *is the restriction of $\leq$ to the set $\mathcal{PE}(P)$;*

(3) $\#_P$ *is the restriction of $\#$ to the set $\mathcal{PE}(P)$.*

In the following we shall feel free to drop the subscript in $\leq_P$ and $\#_P$.

Note that the set $\mathcal{PE}(P)$ may be denumerable, as shown by the following example.

Example 4.4 . *If* $P = \mathsf{q}!\lambda; P \oplus \mathsf{q}!\lambda'$, *then*

$$\mathcal{PE}(P) = \{\underbrace{\mathsf{q}!\lambda \cdot \ldots \cdot \mathsf{q}!\lambda}_{n} \mid n \geq 1\} \cup \{\underbrace{\mathsf{q}!\lambda \cdot \ldots \cdot \mathsf{q}!\lambda}_{n} \cdot \mathsf{q}!\lambda' \mid n \geq 0\}$$

We conclude this section by showing that the ESs of processes are PESs.

Proposition 4.5 . *Let P be a process. Then $\mathcal{S}^{\mathcal{P}}(P)$ is a prime event structure with an empty concurrency relation.*

Proof. We show that $\leq$ and $\#$ satisfy Properties (2) and (3) of Definition 3.1. Reflexivity, transitivity and antisymmetry of $\leq$ follow from the corresponding properties of $\sqsubseteq$. As for irreflexivity and symmetry of $\#$, they follow from Clause (2) of Definition 4.2 and the symmetry of inequality. To show conflict hereditariness, suppose that $\eta \# \eta' \leq \eta''$. From Clause (2) of Definition 4.2 there are π, π', ζ, ζ' and ζ'' such that $\pi \neq \pi'$ and $\eta = \zeta \cdot \pi \cdot \zeta'$ and $\eta' = \zeta \cdot \pi' \cdot \zeta''$. From $\eta' \leq \eta''$ we derive that $\eta'' = \zeta \cdot \pi' \cdot \zeta'' \cdot \zeta_1$ for some ζ_1. Therefore again from Clause (2) we obtain $\eta \# \eta''$. □

5. Event Structure Semantics of Networks

We present now the ES semantics of networks. In the ES of a network, asynchronous communication will be modelled by two causally related events, the first representing an asynchronous output, namely the enqueuing of a message in the queue, and the second representing an asynchronous input, namely the dequeuing of a message from the queue.

We start by defining the *o-trace of a queue* $\mathcal{M}$, notation $\mathsf{otr}(\mathcal{M})$, which is the sequence of output communications in the queue. We use ω to range over o-traces.

Definition 5.1 . *The* o-trace corresponding to a queue *is defined by*

$$\mathsf{otr}(\emptyset) = \epsilon \qquad \mathsf{otr}(\langle \mathsf{p}, \lambda, \mathsf{q} \rangle \cdot \mathcal{M}) = \mathsf{pq}!\lambda \cdot \mathsf{otr}(\mathcal{M})$$

O-traces are considered modulo the following equivalence $\cong$, which mimics the structural equivalence on queues.

Definition 5.2 (o-trace equivalence $\cong$)**.** *The equivalence $\cong$ on o-traces is the least equivalence such that*

$$\omega \cdot \mathsf{pq}!\lambda \cdot \mathsf{rs}!\lambda' \cdot \omega' \;\cong\; \omega \cdot \mathsf{rs}!\lambda' \cdot \mathsf{pq}!\lambda \cdot \omega' \text{ if } \mathsf{p} \neq \mathsf{r} \text{ or } \mathsf{q} \neq \mathsf{s}$$

Network events are relative to a given participant p. They are pairs made of an o-trace ω modulo $\cong$ (notation $\omega_{\cong}$), representing a queue modulo $\equiv$, and of a p-event η.

Definition 5.3 (Network events)**.** (1) Network events ρ, ρ', *also called n-events, are pairs* $(\omega_{\cong}, \eta)$ *located at some participant* p, *written* $\mathsf{p} :: (\omega_{\cong}, \eta)$. *We call* $\omega_{\cong}$ *the* queue *and* η *the p-event of* ρ, *respectively.*

(2) *We define* $\mathsf{i/o}(\rho) = \begin{cases} \mathsf{pq}!\lambda & \text{if } \rho = \mathsf{p} :: (\omega_{\cong}, \zeta \cdot \mathsf{q}!\lambda) \\ \mathsf{pq}?\lambda & \text{if } \rho = \mathsf{q} :: (\omega_{\cong}, \zeta \cdot \mathsf{p}?\lambda) \end{cases}$

and we say that ρ *is an* output n-event *representing the communication* $\mathsf{pq}!\lambda$ *or is an* input n-event *representing the communication* $\mathsf{pq}?\lambda$, *respectively.*

(3) *We denote by* $\mathcal{NE}$ *the set of n-events.*

In order to define the flow relation between an output n-event $\mathsf{p} :: (\omega_{\cong}, \zeta \cdot \mathsf{q}!\lambda)$ and the matching input n-event $\mathsf{q} :: (\omega_{\cong}, \zeta \cdot \mathsf{p}?\lambda)$, we introduce a duality relation on projections of action sequences, see Definition 5.5. We first define the projection of traces on participants, producing action sequences (Definition 5.4(1)), and then the projection of action sequences on participants, producing sequences of *undirected actions* of the form $!\lambda$ and $?\lambda$ (Definition 5.4(2)).

In the sequel, we will use the symbol $\dagger$ to stand for either ! or ?. Then $\mathsf{p}\dagger\lambda$ will stand for either $\mathsf{p}!\lambda$ or $\mathsf{p}?\lambda$. Similarly, $\dagger\lambda$ will stand for either $!\lambda$ or $?\lambda$.

Definition 5.4 (Projections)**.** (1) *The* projection of a trace on a participant *is defined by:*

$$\epsilon @ \mathsf{r} = \epsilon \qquad (\beta \cdot \tau) @ \mathsf{r} = \begin{cases} \mathsf{q}!\lambda \cdot \tau @ \mathsf{r} & \text{if } \beta = \mathsf{rq}!\lambda \\ \mathsf{p}?\lambda \cdot \tau @ \mathsf{r} & \text{if } \beta = \mathsf{pr}?\lambda \\ \tau @ \mathsf{r} & \text{otherwise} \end{cases}$$

(2) *The* projection of an action sequence on a participant *is defined by:*

$$\epsilon \upharpoonright \mathsf{r} = \epsilon \qquad (\pi \cdot \zeta) \upharpoonright \mathsf{r} = \begin{cases} \dagger\lambda \cdot \zeta \upharpoonright \mathsf{r} & \text{if } \pi = \mathsf{r}\dagger\lambda \\ \zeta \upharpoonright \mathsf{r} & \text{otherwise} \end{cases}$$

We use χ to range over sequences of output actions and ϑ to range over sequences of undirected actions.

We now introduce a partial order relation $\lessapprox$ on sequences of undirected actions, which reflects the fact that in an asynchronous semantics it is better to anticipate outputs, as first observed in [MYH09]. This relation, as well as the standard duality relation $\bowtie$ on projections, will be used to define our specific duality relation ${}_{\lessapprox}\bowtie_{\gtrsim}$ on projections of action sequences.

Definition 5.5 (Partial order and duality relations on undirected action sequences)**.** *The three relations $\lessapprox$, $\bowtie$ and ${}_{\lessapprox}\bowtie_{\gtrsim}$ on undirected action sequences are defined as follows:*

(1) *The relation $\lessapprox$ on undirected action sequences is defined as the smallest partial order such that:*

$$\vartheta \cdot !\lambda \cdot ?\lambda' \cdot \vartheta' \lessapprox \vartheta \cdot ?\lambda' \cdot !\lambda \cdot \vartheta'$$

(2) *The relation $\bowtie$ on undirected action sequences is defined by:*

$$\epsilon \bowtie \epsilon \qquad \vartheta \bowtie \vartheta' \quad \Rightarrow \quad !\lambda.\vartheta \bowtie ?\lambda.\vartheta' \text{ and } ?\lambda.\vartheta \bowtie !\lambda.\vartheta'$$

(3) *The relation ${}_{\lessapprox}\bowtie_{\gtrsim}$ undirected action sequences is defined by:*

$$\vartheta_1 \lessapprox\bowtie_{\gtrsim} \vartheta_2 \qquad \text{if} \qquad \vartheta_1' \bowtie \vartheta_2' \text{ for some } \vartheta_1', \vartheta_2' \text{ such that } \vartheta_1 \lessapprox \vartheta_1' \text{ and } \vartheta_2 \lessapprox \vartheta_2'$$

For example $!\lambda_1 \cdot !\lambda_2 \cdot ?\lambda_3 \lessapprox ?\lambda_3 \cdot !\lambda_1 \cdot !\lambda_2$, which implies $!\lambda_1 \cdot !\lambda_2 \cdot ?\lambda_3 \;{}_{\lessapprox}\bowtie_{\gtrsim}\; !\lambda_3 \cdot ?\lambda_1 \cdot ?\lambda_2$.

We may now define the flow and conflict relations on n-events.

Definition 5.6 (Flow and conflict relations on n-events)**.** *The* flow *relation $\prec$ and the* conflict *relation $\#$ on the set of n-events $\mathcal{NE}$ are defined by:*

(1) (a) $\eta < \eta' \Rightarrow \mathsf{p} :: (\omega_{\cong}, \eta) \prec \mathsf{p} :: (\omega_{\cong}, \eta')$;
(b) $(\omega @ \mathsf{p} \cdot \zeta) \upharpoonright \mathsf{q} \;{}_{\lessapprox}\bowtie_{\gtrsim}\; (\omega @ \mathsf{q} \cdot \zeta'') \upharpoonright \mathsf{p}$ *and* $(\zeta' \cdot \mathsf{p}?\lambda) \upharpoonright \mathsf{p} \lessapprox (\zeta'' \cdot \mathsf{p}?\lambda \cdot \chi) \upharpoonright \mathsf{p}$ *for some ζ'' and* $\chi \Rightarrow \mathsf{p} :: (\omega_{\cong}, \zeta \cdot \mathsf{q}!\lambda) \prec \mathsf{q} :: (\omega_{\cong}, \zeta' \cdot \mathsf{p}?\lambda)$;

(2) $\eta \# \eta' \Rightarrow \mathsf{p} :: (\omega_{\cong}, \eta) \# \mathsf{p} :: (\omega_{\cong}, \eta')$.

Clause (1a) defines flows within the same "locality" p, which we call *local flows*, while Clause (1b) defines flows between different localities, which we call *cross-flows*: these are flows between an output of p towards q and the corresponding input of q from p. The condition in this clause expresses a sort of "weak duality" between the history of the output and the history of the input: the intuition is that if q has some outputs towards p occurring

in ζ', namely before its input $\mathsf{p}?\lambda$, then when checking for duality these outputs can be moved after $\mathsf{p}?\lambda$, namely in χ, because q does not need to wait until p has consumed these outputs to perform its input $\mathsf{p}?\lambda$. This condition can be seen at work in Examples 5.10 and 5.11.

The reason for using the same o-traces when defining the flow and conflict relations is that in the ES of a network they will be images through otr of the same queue.

For example, we have a cross-flow $\rho \prec \rho'$ between the following n-events, where $\omega = \mathsf{pq}!\lambda_1 \cdot \mathsf{pq}!\lambda_2 \cdot \mathsf{qs}!\lambda_5 \cdot \mathsf{qp}!\lambda_3$:

$$\rho = \mathsf{p} :: (\omega_{\cong}, \mathsf{r}?\lambda_4 \cdot \mathsf{q}?\lambda_3 \cdot \mathsf{q}!\lambda) \prec \mathsf{q} :: (\omega_{\cong}, \mathsf{p}!\lambda' \cdot \mathsf{p}?\lambda_1 \cdot \mathsf{p}?\lambda_2 \cdot \mathsf{p}?\lambda) = \rho'$$

since in this case $\zeta = \mathsf{r}?\lambda_4 \cdot \mathsf{q}?\lambda_3$ and $\zeta' = \mathsf{p}!\lambda' \cdot \mathsf{p}?\lambda_1 \cdot \mathsf{p}?\lambda_2$, and thus, taking $\zeta'' = \mathsf{p}?\lambda_1 \cdot \mathsf{p}?\lambda_2$ and $\chi = \mathsf{p}!\lambda'$, we obtain

$$(\omega \mathbin{@} \mathsf{p} \cdot \zeta) \Rsh \mathsf{q} \;=\; !\lambda_1 \cdot !\lambda_2 \cdot ?\lambda_3 \lessapprox \; ?\lambda_3 \cdot !\lambda_1 \cdot !\lambda_2 \;\bowtie\; !\lambda_3 \cdot ?\lambda_1 \cdot ?\lambda_2 = (\omega \mathbin{@} \mathsf{q} \cdot \zeta'') \Rsh \mathsf{p}$$

and

$$(\zeta' \cdot \mathsf{p}?\lambda) \Rsh \mathsf{p} \;=\; !\lambda' \cdot ?\lambda_1 \cdot ?\lambda_2 \cdot ?\lambda \lessapprox \; ?\lambda_1 \cdot ?\lambda_2 \cdot ?\lambda \cdot !\lambda' = (\zeta'' \cdot \mathsf{p}?\lambda \cdot \chi) \Rsh \mathsf{p}$$

When $\rho = \mathsf{p} :: (\omega_{\cong}, \eta) \prec \mathsf{q} :: (\omega_{\cong}, \eta') = \rho'$ and $\mathsf{p} \neq \mathsf{q}$, then by definition ρ is an output and ρ' is an input. In this case we say that the output ρ *justifies* the input ρ', or symmetrically that the input ρ' *is justified* by the output ρ.

An input n-event may also be justified by a message in the queue. This is formalised by the following definition.

Definition 5.7 (Queue-justified n-events)**.** *The input n-event* $\rho = \mathsf{q} :: ((\omega \cdot \mathsf{pq}!\lambda \cdot \omega')_{\cong}, \zeta \cdot \mathsf{p}?\lambda)$ *is* queue-justified *if* $(\omega \mathbin{@} \mathsf{p}) \Rsh \mathsf{q} \lessapprox \bowtie_{\gtrsim} \zeta' \Rsh \mathsf{p}$ *and* $(\zeta \cdot \mathsf{p}?\lambda) \Rsh \mathsf{p} \lessapprox (\zeta' \cdot \mathsf{p}?\lambda \cdot \chi) \Rsh \mathsf{p}$ *for some* ζ' *and* χ*.*

The conditions $(\omega \mathbin{@} \mathsf{p}) \Rsh \mathsf{q} \lessapprox \bowtie_{\gtrsim} \zeta' \Rsh \mathsf{p}$ and $(\zeta \cdot \mathsf{p}?\lambda) \Rsh \mathsf{p} \lessapprox (\zeta' \cdot \mathsf{p}?\lambda \cdot \chi) \Rsh \mathsf{p}$ for some ζ' and χ ensure that the input will consume the shown message in the queue. For example, if $\omega = \mathsf{pq}!\lambda \cdot \mathsf{pq}!\lambda$, then both $\mathsf{q} :: (\omega_{\cong}, \mathsf{p}!\lambda' \cdot \mathsf{p}?\lambda)$ and $\mathsf{q} :: (\omega_{\cong}, \mathsf{p}!\lambda' \cdot \mathsf{p}?\lambda \cdot \mathsf{p}?\lambda)$ are queue-justified. On the other hand, if $\omega = \mathsf{pq}!\lambda$ then $\mathsf{q} :: (\omega_{\cong}, \mathsf{p}?\lambda \cdot \mathsf{p}?\lambda)$ is not queue-justified.

To define the set of n-events associated with a network, we filter the set of all its potential n-events by keeping only

- those n-events whose constituent p-events have all their predecessors appearing in some other n-event of the network and
- those input n-events that are either queue justified or justified by output n-events of the network.

Given a set E of n-events, we define the *narrowing* of E (notation $\mathsf{nr}(E)$) as the greatest fixpoint of the function f_E on sets of n-events defined by:

$$\begin{aligned} f_E(X) \;=\; & \{\rho \in E \mid \rho = \mathsf{p} :: (\omega_{\cong}, \eta \cdot \pi) \;\Rightarrow\; \exists \rho' \in X \,.\, \rho' = \mathsf{p} :: (\omega_{\cong}, \eta) \text{ and} \\ & (\,\rho \text{ is an input n-event} \Rightarrow \rho \text{ is either queue-justified or justified by some } \rho'' \in X\,)\} \end{aligned}$$

Thus, $\mathsf{nr}(E)$ is the greatest set $X \subseteq E$ such that $X = f_E(X)$.

Note that we could not have taken $\mathsf{nr}(E)$ to be the least fixpoint of f_E rather than its greatest fixpoint. Indeed, the least fixpoint of f_E would be the empty set.

We have now enough machinery to define the ES of networks.

Definition 5.8 (Event Structure of a network)**.** *The* event structure of network $\mathsf{N} \parallel \mathcal{M}$ *is the triple:*

$$\mathcal{S}^{\mathcal{N}}(\mathsf{N} \parallel \mathcal{M}) = (\mathcal{NE}(\mathsf{N} \parallel \mathcal{M}), \prec, \#)$$

where

(1) $\mathcal{NE}(\mathsf{N} \parallel \mathcal{M}) = \mathsf{nr}(\{\mathsf{p} :: (\omega_{\cong}, \eta) \mid \omega = \mathsf{otr}(\mathcal{M})$ *and* $\eta \in \mathcal{PE}(P)$ *with* $\mathsf{p}[\![\, P \,]\!] \in \mathsf{N}\})$;
(2) $\prec_{\mathsf{N} \parallel \mathcal{M}}$ *is the restriction of* $\prec$ *to the set* $\mathcal{NE}(\mathsf{N} \parallel \mathcal{M})$;
(3) $\#_{\mathsf{N} \parallel \mathcal{M}}$ *is the restriction of* $\#$ *to the set* $\mathcal{NE}(\mathsf{N} \parallel \mathcal{M})$.

The following example shows how the operation of narrowing prunes the set of potential n-events of a network ES. It also illustrates the interplay between the two conditions in the definition of narrowing.

Example 5.9 . *Consider the network* $\mathsf{N} \parallel \emptyset$, *where* $\mathsf{N} = \mathsf{p}[\![\, \mathsf{q}?\lambda; \mathsf{r}!\lambda' \,]\!] \parallel \mathsf{r}[\![\, \mathsf{p}?\lambda' \,]\!]$. *The set of potential n-events of* $\mathcal{S}^{\mathcal{N}}(\mathsf{N} \parallel \emptyset)$ *is* $\{\mathsf{p} :: (\epsilon, \mathsf{q}?\lambda), \mathsf{p} :: (\epsilon, \mathsf{q}?\lambda; \mathsf{r}!\lambda'), \mathsf{r} :: (\epsilon, \mathsf{p}?\lambda')\}$. *The n-event* $\mathsf{p} :: (\epsilon, \mathsf{q}?\lambda)$ *is cancelled, since it is neither queue-justified nor justified by another n-event of the ES. Then* $\mathsf{p} :: (\epsilon, \mathsf{q}?\lambda; \mathsf{r}!\lambda')$ *is cancelled since it lacks its predecessor* $\mathsf{p} :: (\epsilon, \mathsf{q}?\lambda)$. *Lastly* $\mathsf{r} :: (\epsilon, \mathsf{p}?\lambda')$ *is cancelled, since it is neither queue-justified nor justified by another n-event of the ES. Notice that* $\mathsf{p} :: (\epsilon, \mathsf{q}?\lambda; \mathsf{r}!\lambda')$ *would have justified* $\mathsf{r} :: (\epsilon, \mathsf{p}?\lambda')$, *if it had not been cancelled. We conclude that* $\mathcal{NE}(\mathsf{N} \parallel \emptyset) = \emptyset$.

Example 5.10 . *Consider the ES associated with the network* $\mathsf{N} \parallel \emptyset$, *with*

$$\mathsf{N} = \mathsf{p}[\![\, \mathsf{q}!\lambda; \mathsf{q}?\lambda'; \mathsf{q}!\lambda; \mathsf{q}?\lambda' \,]\!] \parallel \mathsf{q}[\![\, \mathsf{p}!\lambda'; \mathsf{p}?\lambda; \mathsf{p}!\lambda'; \mathsf{p}?\lambda \,]\!]$$

The n-events of $\mathcal{S}^{\mathcal{N}}(\mathsf{N} \parallel \emptyset)$ *are:*

$$\begin{array}{ll}
\rho_1 = \mathsf{p} :: (\epsilon, \mathsf{q}!\lambda) & \rho'_1 = \mathsf{q} :: (\epsilon, \mathsf{p}!\lambda') \\
\rho_2 = \mathsf{p} :: (\epsilon, \mathsf{q}!\lambda \cdot \mathsf{q}?\lambda') & \rho'_2 = \mathsf{q} :: (\epsilon, \mathsf{p}!\lambda' \cdot \mathsf{p}?\lambda) \\
\rho_3 = \mathsf{p} :: (\epsilon, \mathsf{q}!\lambda \cdot \mathsf{q}?\lambda' \cdot \mathsf{q}!\lambda) & \rho'_3 = \mathsf{q} :: (\epsilon, \mathsf{p}!\lambda' \cdot \mathsf{p}?\lambda \cdot \mathsf{p}!\lambda') \\
\rho_4 = \mathsf{p} :: (\epsilon, \mathsf{q}!\lambda \cdot \mathsf{q}?\lambda' \cdot \mathsf{q}!\lambda \cdot \mathsf{q}?\lambda') & \rho'_4 = \mathsf{q} :: (\epsilon, \mathsf{p}!\lambda' \cdot \mathsf{p}?\lambda \cdot \mathsf{p}!\lambda' \cdot \mathsf{p}?\lambda)
\end{array}$$

The flow relation is given by the cross-flows $\rho_1 \prec \rho'_2, \rho_3 \prec \rho'_4, \rho'_1 \prec \rho_2, \rho'_3 \prec \rho_4$, *as well as by the local flows* $\rho_i \prec \rho_j$ *and* $\rho'_i \prec \rho'_j$ *for all* i, j *such that* $i \in \{1, 2, 3\}$, $j \in \{2, 3, 4\}$ *and* $i < j$. *The conflict relation is empty.*

The configurations of $\mathcal{S}^{\mathcal{N}}(\mathsf{N} \parallel \emptyset)$ *are:*

$$\begin{array}{llllll}
\{\rho_1\} & \{\rho'_1\} & \{\rho_1, \rho'_1\} & \{\rho_1, \rho'_1, \rho_2\} & \{\rho_1, \rho'_1, \rho'_2\} & \{\rho_1, \rho'_1, \rho_2, \rho'_2\} \\
\{\rho_1, \rho'_1, \rho_2, \rho_3\} & \{\rho_1, \rho'_1, \rho'_2, \rho'_3\} & \{\rho_1, \rho'_1, \rho_2, \rho'_2, \rho_3\} & \{\rho_1, \rho'_1, \rho_2, \rho'_2, \rho'_3\} & \{\rho_1, \rho'_1, \rho_2, \rho'_2, \rho_3, \rho'_3\} & \\
\{\rho_1, \rho'_1, \rho_2, \rho'_2, \rho_3, \rho'_3, \rho_4\} & \{\rho_1, \rho'_1, \rho_2, \rho'_2, \rho_3, \rho'_3, \rho'_4\} & \{\rho_1, \rho'_1, \rho_2, \rho'_2, \rho_3, \rho'_3, \rho_4, \rho'_4\} & & &
\end{array}$$

The network $\mathsf{N} \parallel \emptyset$ *can evolve in two steps to the network:*

$$\mathsf{N}' \parallel \mathcal{M}' = \mathsf{p}[\![\, \mathsf{q}?\lambda'; \mathsf{q}!\lambda; \mathsf{q}?\lambda' \,]\!] \parallel \mathsf{q}[\![\, \mathsf{p}?\lambda; \mathsf{p}!\lambda'; \mathsf{p}?\lambda \,]\!] \parallel \langle \mathsf{p}, \lambda, \mathsf{q} \rangle \cdot \langle \mathsf{q}, \lambda', \mathsf{p} \rangle$$

The n-events of $\mathcal{S}^{\mathcal{N}}(\mathsf{N}' \parallel \mathcal{M}')$ *are:*

$$\begin{array}{ll}
\rho_5 = \mathsf{p} :: (\omega_{\cong}, \mathsf{q}?\lambda') & \rho'_5 = \mathsf{q} :: (\omega_{\cong}, \mathsf{p}?\lambda) \\
\rho_6 = \mathsf{p} :: (\omega_{\cong}, \mathsf{q}?\lambda' \cdot \mathsf{q}!\lambda) & \rho'_6 = \mathsf{q} :: (\omega_{\cong}, \mathsf{p}?\lambda \cdot \mathsf{p}!\lambda') \\
\rho_7 = \mathsf{p} :: (\omega_{\cong}, \mathsf{q}?\lambda' \cdot \mathsf{q}!\lambda \cdot \mathsf{q}?\lambda') & \rho'_7 = \mathsf{q} :: (\omega_{\cong}, \mathsf{p}?\lambda \cdot \mathsf{p}!\lambda' \cdot \mathsf{p}?\lambda)
\end{array}$$

where $\omega = \mathsf{pq}!\lambda \cdot \mathsf{qp}!\lambda'$. *The flow relation is given by the cross-flows* $\rho_6 \prec \rho'_7$, $\rho'_6 \prec \rho_7$, *and by the local flows* $\rho_i \prec \rho_j$ *and* $\rho'_i \prec \rho'_j$ *for all* i, j *such that* $i \in \{5, 6\}$, $j \in \{6, 7\}$ *and* $i < j$. *The input n-events* ρ_5 *and* ρ'_5, *which are the only ones without causes, are queue-justified. The conflict relation is empty.*

The network $\mathsf{N}' \parallel \mathcal{M}'$ *can evolve in five steps to the network:*

$$\mathsf{N}'' \parallel \mathcal{M}'' = \mathsf{q}[\![\, \mathsf{p}?\lambda \,]\!] \parallel \langle \mathsf{p}, \lambda, \mathsf{q} \rangle$$

The only n-event of $\mathcal{S}^{\mathcal{N}}(\mathsf{N}'' \parallel \mathcal{M}'')$ *is* $\mathsf{q} :: (\mathsf{pq}!\lambda, \mathsf{p}?\lambda)$.

Example 5.11 . *Let* $\mathsf{N} = \mathsf{p}[\![\, \mathsf{q}!\lambda_1; \mathsf{r}!\lambda \oplus \mathsf{q}!\lambda_2; \mathsf{r}!\lambda \,]\!] \parallel \mathsf{q}[\![\, \mathsf{p}?\lambda_1 + \mathsf{p}?\lambda_2 \,]\!] \parallel \mathsf{r}[\![\, \mathsf{p}?\lambda \,]\!]$. *The n-events of* $\mathcal{S}^{\mathcal{N}}(\mathsf{N} \parallel \emptyset)$ *are:*

$$\rho_1 = \mathsf{p} :: (\epsilon, \mathsf{q}!\lambda_1) \qquad \rho_1' = \mathsf{q} :: (\epsilon, \mathsf{p}?\lambda_1)$$
$$\rho_2 = \mathsf{p} :: (\epsilon, \mathsf{q}!\lambda_1 \cdot \mathsf{r}!\lambda) \qquad \rho_2' = \mathsf{q} :: (\epsilon, \mathsf{p}?\lambda_2)$$
$$\rho_3 = \mathsf{p} :: (\epsilon, \mathsf{q}!\lambda_2) \qquad \rho_1'' = \mathsf{r} :: (\epsilon, \mathsf{p}?\lambda)$$
$$\rho_4 = \mathsf{p} :: (\epsilon, \mathsf{q}!\lambda_2 \cdot \mathsf{r}!\lambda)$$

The flow relation is given by the local flows $\rho_1 \prec \rho_2$, $\rho_3 \prec \rho_4$, *and by the cross-flows* $\rho_1 \prec \rho_1'$, $\rho_2 \prec \rho_1''$, $\rho_3 \prec \rho_2'$, $\rho_4 \prec \rho_1''$. *The conflict relation is given by* $\rho_1 \# \rho_3$, $\rho_1 \# \rho_4$, $\rho_2 \# \rho_3$, $\rho_2 \# \rho_4$ *and* $\rho_1' \# \rho_2'$. *Notice that* ρ_2 *and* ρ_4 *are conflicting causes of* ρ_1''. *The configurations are*

$$\{\rho_1\} \quad \{\rho_1, \rho_2\} \quad \{\rho_1, \rho_1'\} \quad \{\rho_1, \rho_2, \rho_1'\} \quad \{\rho_1, \rho_2, \rho_1''\} \quad \{\rho_1, \rho_2, \rho_1', \rho_1''\}$$
$$\{\rho_3\} \quad \{\rho_3, \rho_4\} \quad \{\rho_3, \rho_2'\} \quad \{\rho_3, \rho_4, \rho_2'\} \quad \{\rho_3, \rho_4, \rho_1''\} \quad \{\rho_3, \rho_4, \rho_2', \rho_1''\}$$

The network $\mathsf{N} \parallel \mathcal{M}$ *can evolve in one step to the network:*

$$\mathsf{N}' \parallel \mathcal{M}' = \mathsf{p}[\![\, \mathsf{r}!\lambda \,]\!] \parallel \mathsf{q}[\![\, \mathsf{p}?\lambda_1 + \mathsf{p}?\lambda_2 \,]\!] \parallel \mathsf{r}[\![\, \mathsf{p}?\lambda \,]\!] \parallel \langle \mathsf{p}, \lambda_1, \mathsf{q} \rangle$$

The n-events of $\mathcal{S}^{\mathcal{N}}(\mathsf{N}' \parallel \mathcal{M}')$ *are* $\rho_5 = \mathsf{p} :: (\omega, \mathsf{r}!\lambda)$, $\rho_3' = \mathsf{q} :: (\omega, \mathsf{p}?\lambda_1)$ *and* $\rho_2'' = \mathsf{r} :: (\omega, \mathsf{p}?\lambda)$, *where* $\omega = \mathsf{pq}!\lambda_1$. *The flow relation is given by the cross-flow* $\rho_5 \prec \rho_2''$. *Notice that the input n-event* ρ_3' *is queue-justified, and that there is no n-event corresponding to the branch* $\mathsf{p}?\lambda_2$ *of* q, *since such an n-event would not be queue-justified. Hence the conflict relation is empty. The configurations are*

$$\{\rho_5\} \quad \{\rho_3'\} \quad \{\rho_5, \rho_3'\} \quad \{\rho_5, \rho_2''\} \quad \{\rho_5, \rho_3', \rho_2''\}$$

It is easy to show that the ESs of networks are FESs.

Proposition 5.12 . *Let* $\mathsf{N} \parallel \mathcal{M}$ *be a network. Then* $\mathcal{S}^{\mathcal{N}}(\mathsf{N} \parallel \mathcal{M})$ *is a flow event structure.*

Proof. The relation $\prec$ is irreflexive since:

(1) $\eta < \eta'$ implies $\mathsf{p} :: (\omega_{\cong}, \eta) \neq \mathsf{p} :: (\omega_{\cong}, \eta')$;

(2) $\mathsf{p} \neq \mathsf{q}$ implies $\mathsf{p} :: (\omega_{\cong}, \zeta \cdot \mathsf{q}!\lambda) \neq \mathsf{q} :: (\omega_{\cong}, \zeta' \cdot \mathsf{p}?\lambda)$.

Symmetry of the conflict relation between n-events follows from the corresponding property of conflict between p-events. □

In the remainder of this section we show that projections of n-event configurations give p-event configurations. We start by formalising the projection function of n-events to p-events and showing that it is downward surjective.

Definition 5.13 (Projection of n-events to p-events)**.** *The projection function* $proj_{\mathsf{p}}(\cdot)$ *is defined by:*

$$proj_{\mathsf{p}}(\rho) = \begin{cases} \eta & \text{if } \rho = \mathsf{p} :: (\omega_{\cong}, \eta) \\ \textit{undefined} & \textit{otherwise} \end{cases}$$

The projection function $proj_{\mathsf{p}}(\cdot)$ *is extended to sets of n-events in the obvious way:*

$$proj_{\mathsf{p}}(X) = \{\eta \mid \exists \rho \in X \,.\, proj_{\mathsf{p}}(\rho) = \eta\}$$

Proposition 5.14 (Downward surjectivity of projections)**.** *Let*

$$\mathsf{p}[\![\, P \,]\!] \in \mathsf{N},\ \mathcal{S}^{\mathcal{N}}(\mathsf{N} \parallel \mathcal{M}) = (\mathcal{NE}(\mathsf{N} \parallel \mathcal{M}), \prec, \#) \text{ and } \mathcal{S}^{\mathcal{P}}(P) = (\mathcal{PE}(P), \leq_P, \#_P)$$

Then the partial function $proj_{\mathsf{p}} : \mathcal{NE}(\mathsf{N} \parallel \mathcal{M}) \to \mathcal{PE}(P)$ *is downward surjective.*

Proof. Follows immediately from the fact that $\mathcal{NE}(\mathsf{N} \parallel \mathcal{M})$ is the narrowing of a set of n-events $\mathsf{p} :: (\omega_{\cong}, \eta)$ with $\omega = \mathsf{otr}(\mathcal{M})$ and $\mathsf{p}[\![\, P \,]\!] \in \mathsf{N}$ and $\eta \in \mathcal{PE}(P)$. □

The operation of narrowing on network events makes sure that each configuration of the ES of a network projects down to configurations of the ESs of the component processes.

Proposition 5.15 (Projection preserves configurations)**.** *Let* $\mathsf{p}[\![\, P \,]\!] \in \mathsf{N}$*. If* $\mathcal{X} \in \mathcal{C}(\mathcal{S}^{\mathcal{N}}(\mathsf{N} \parallel \mathcal{M}))$*, then* $proj_{\mathsf{p}}(\mathcal{X}) \in \mathcal{C}(\mathcal{S}^{\mathcal{P}}(P))$*.*

Proof. Let $\mathcal{X} \in \mathcal{C}(\mathcal{S}^{\mathcal{N}}(\mathsf{N} \parallel \mathcal{M}))$ and $\mathcal{Y} = proj_{\mathsf{p}}(\mathcal{X})$. We want to show that $\mathcal{Y} \in \mathcal{C}(\mathcal{S}^{\mathcal{P}}(P))$, namely that $\mathcal{Y}$ satisfies Conditions (1) and (2) of Definition 3.3.

(1) *Downward-closure.* Let $\eta \in \mathcal{Y}$. Since $\mathcal{Y} = proj_{\mathsf{p}}(\mathcal{X})$, there exists $\rho \in \mathcal{X}$ such that $\rho = \mathsf{p} :: (\omega_{\cong}, \eta)$. Suppose $\eta' < \eta$. From Proposition 5.14 there exists $\rho' \in \mathcal{NE}(\mathsf{N} \parallel \mathcal{M})$ such that $\rho' = \mathsf{p} :: (\omega_{\cong}, \eta')$. By Definition 5.8(1) we have then $\rho' \prec \rho$. Since $\mathcal{X}$ is left-closed up to conflicts, we know that either $\rho' \in \mathcal{X}$ or there exists $\rho'' \in \mathcal{X}$ such that $\rho'' \# \rho'$ and $\rho'' \prec \rho$. We examine the two cases in turn:
   – $\rho' \in \mathcal{X}$. Then, since $\eta' = proj_{\mathsf{p}}(\rho')$, we have $\eta' \in proj_{\mathsf{p}}(\mathcal{X}) = \mathcal{Y}$ and we are done.
   – $\exists \rho'' \in \mathcal{X}\,.\ \rho'' \# \rho'$ and $\rho'' \prec \rho$. From $\rho'' \# \rho'$ we get $\rho'' = \mathsf{p} :: (\omega_{\cong}, \eta'')$ and $\eta'' \# \eta'$. This implies $\eta'' \# \eta$. By Definition 5.8(2) this implies $\rho \# \rho'$, contradicting the hypothesis that $\mathcal{X}$ is conflict-free. So this case is impossible.

(2) *Conflict-freeness.* Ad absurdum, suppose there exist $\eta, \eta' \in \mathcal{Y}$ such that $\eta \# \eta'$. Then, since $\mathcal{Y} = proj_{\mathsf{p}}(\mathcal{X})$, there must exist $\rho, \rho' \in \mathcal{X}$ such that $\rho = \mathsf{p} :: (\omega_{\cong}, \eta)$ and $\rho' = \mathsf{p} :: (\omega_{\cong}, \eta')$. By Definition 5.8(2) this implies $\rho \# \rho'$, contradicting the hypothesis that $\mathcal{X}$ is conflict-free. □

6. Asynchronous Global Types

In this section we introduce our new global types for asynchronous communication. The underlying idea is quite simple: to split the communication constructor of standard global types into an output constructor and an input constructor. This will allow us to type networks in which all participants make all their outputs before their inputs, like the network of Example 2.4, whose asynchronous global types will be presented in Example 6.8.

Definition 6.1 (Global types)**.** (1) Pre-sequential global types *(pre-sgts) are defined coinductively by:*

$$\mathsf{G} \quad ::=^{coind} \quad \mathsf{pq!} \boxplus_{i \in I} \lambda_i ; \mathsf{G}_i \ \mid\ \mathsf{pq?} \lambda ; \mathsf{G} \ \mid\ \mathsf{End}$$

where I is non-empty and $\lambda_j \neq \lambda_h$ for all $j, h \in I$, $j \neq h$, i.e. labels in choices are all different. The pre-sgt $\mathsf{pq!} \boxplus_{i \in I} \lambda_i ; \mathsf{G}_i$ *specifies a send from* p *to* q *of one of the labels* λ_i*, followed by the behaviour specified by* G_i*. Dually, the pre-sgt* $\mathsf{pq?} \lambda ; \mathsf{G}$ *specifies a read by* q *of the label* λ *sent by* p*, followed by the behaviour specified by* G*.*

(2) *The* tree representation of a pre-sgt *is a directed rooted tree, where: (a) each internal node represents either a choice* $\mathsf{pq!} \boxplus_{i \in I} \lambda_i ; \mathsf{G}_i$*, in which case it is decorated by* $\mathsf{pq!}$ *and has as many children as there are branches in the choice, or an input* $\mathsf{pq?} \lambda ; \mathsf{G}$*, in which case it is decorated by* $\mathsf{pq?}$ *and has a unique child, and (b) the edge from* $\mathsf{pq!}$ *to the child* G_i *is decorated by* λ_i *and the one from* $\mathsf{pq?}$ *to the child* G *is decorated by* λ*, and (c) the leaves of the tree (if any) are decorated by* End*. The tree representation of* End *has a unique node which is a leaf.*

(3) *We say that a pre-sgt* G *is a* sequential global type (sgt) *if the tree representation of* G *is regular (namely, it has finitely many distinct sub-trees).*

(4) Parallel global types (pgts) *are defined by:*

$$\mathcal{G} ::=^{ind} \mathsf{G} \ \mid\ \mathcal{G} \parallel \mathcal{G}$$

where G *is an sgt.*

(5) Asynchronous global types (agts) *are pairs made of a pgt and a queue, written* $\mathcal{G} \parallel \mathcal{M}$*.*

(6) *An asynchronous global type is* simple *if its pgt is an sgt, namely if it has the form* $\mathsf{G} \parallel \mathcal{M}$*.*

$$\mathsf{G} \upharpoonright \mathsf{r} = \mathbf{0} \text{ if } \mathsf{r} \notin \mathsf{play}(\mathsf{G})$$

$$(\mathsf{pq}! \boxplus_{i\in I} \lambda_i; \mathsf{G}_i) \upharpoonright \mathsf{r} = \begin{cases} \bigoplus_{i\in I} \mathsf{q}!\lambda_i; \mathsf{G}_i \upharpoonright \mathsf{p} & \text{if } \mathsf{r} = \mathsf{p}, \\ \mathsf{G}_1 \upharpoonright \mathsf{q} & \text{if } \mathsf{r} = \mathsf{q} \text{ and } |I| = 1 \\ \overrightarrow{\pi}; \Sigma_{i\in I} \mathsf{p}?\lambda_i; P_i & \text{if } \mathsf{r} = \mathsf{q} \text{ and } |I| > 1 \text{ and } \mathsf{G}_i \upharpoonright \mathsf{q} = \overrightarrow{\pi}; \mathsf{p}?\lambda_i; P_i, \\ \mathsf{G}_1 \upharpoonright \mathsf{r} & \text{if } \mathsf{r} \notin \{\mathsf{p}, \mathsf{q}\} \text{ and } \mathsf{r} \in \mathsf{play}(\mathsf{G}_1) \text{ and } \mathsf{G}_i \upharpoonright \mathsf{r} = \mathsf{G}_1 \upharpoonright \mathsf{r} \text{ for all } i \in I \end{cases}$$

$$(\mathsf{pq}?\lambda; \mathsf{G}) \upharpoonright \mathsf{r} = \begin{cases} \mathsf{p}?\lambda; \mathsf{G} \upharpoonright \mathsf{r} & \text{if } \mathsf{r} = \mathsf{q} \\ \mathsf{G} \upharpoonright \mathsf{r} & \text{if } \mathsf{r} \neq \mathsf{q} \text{ and } \mathsf{r} \in \mathsf{play}(\mathsf{G}) \end{cases}$$

$$(\mathcal{G}_1 \parallel \mathcal{G}_2) \upharpoonright \mathsf{r} = \mathcal{G}_i \upharpoonright \mathsf{r} \text{ if } \mathsf{r} \text{ does not occur in } \mathcal{G}_j \text{ for } \{i, j\} = \{1, 2\}$$

Figure 2: Projection of pgts onto participants.

Given an sgt G, the sequences of decorations of nodes and edges on the path from the root to an edge of the tree of G are traces, in the sense of Definition 2.3. We denote by $\mathsf{Tr}^+(\mathsf{G})$ the set of these traces. By definition, $\mathsf{Tr}^+(\mathsf{End}) = \emptyset$ and each trace in $\mathsf{Tr}^+(\mathsf{G})$ is non-empty.

As may be expected, networks will be typed by agts, see Figure 4. A standard guarantee that should be ensured by agts is that each participant whose behaviour is not terminated can do some action. Moreover, since communications are split into inputs and outputs in the syntax of agts, we must make sure that each input has a matching output in the type, and vice versa. To account for all these requirements we will impose a well-formedness condition on agts.

We start by defining the projections of pgts onto participants (Figure 2). We proceed by defining the depth of participants in sgts (Definition 6.2) and the input/output matching predicate (Figure 3). We can then present the typing rules (Figure 4). For establishing the expected properties of the type system we introduce an LTS for simple agts (Figure 5) and show that well-formedness of simple agts is preserved by transitions (Lemma 6.14).

This section is divided in two subsections, the first focussing on well-formedness and the second presenting the type system and showing that it enjoys the properties of subject reduction and session fidelity and that moreover it ensures progress.

6.1. **Well-formed Global Types.**

We start by formalising the set of players of pgts, which will be largely used in the definitions and results presented in this section.

The set of *players of a pgt* $\mathcal{G}$, $\mathsf{play}(\mathcal{G})$, is the smallest set such that:

$$\mathsf{play}(\mathsf{pq}! \boxplus_{i\in I} \lambda_i; \mathsf{G}_i) = \{\mathsf{p}\} \cup \bigcup_{i\in I} \mathsf{play}(\mathsf{G}_i)$$
$$\mathsf{play}(\mathsf{pq}?\lambda; \mathsf{G}) = \{\mathsf{q}\} \cup \mathsf{play}(\mathsf{G})$$
$$\mathsf{play}(\mathsf{End}) = \emptyset$$
$$\mathsf{play}(\mathcal{G} \parallel \mathcal{G}') = \mathsf{play}(\mathcal{G}) \cup \mathsf{play}(\mathcal{G}')$$

The regularity assumption ensures that the set of players of a pgt is finite.

As mentioned earlier, the projection of pgts on participants yields processes. Its coinductive definition is given in Figure 2, where we use $\overrightarrow{\pi}$ to denote any sequence, possibly empty, of input/output actions separated by ";" (note the difference with the sequences ζ

defined after Definition 4.1, where actions are separated by "·").
The projection of an sgt on a participant which is not a player of the type is the inactive process $\mathbf{0}$. In particular, the projection of End is $\mathbf{0}$ on all participants.
The projection of an output choice type on the sender produces an output process sending one of its possible labels to the receiver and then acting according to the projection of the corresponding branch.
The projection of an output choice type on the receiver q has two clauses: one for the case where the choice has a single branch, and one for the case where the choice has more than one branch. In the first case, the projection is simply the projection of the continuation of the single branch on q. In the second case, the projection is defined if the projection of the continuation of each branch on q starts with the same sequence of actions $\overrightarrow{\pi}$, followed by an input of the label sent by p on that branch and then by a possibly different process in each branch. In fact, participant q must receive the label chosen by participant p before behaving differently in different branches. The projection on q is then the initial sequence of actions $\overrightarrow{\pi}$ followed by an external choice on the different sent labels. The sequence $\overrightarrow{\pi}$ is allowed to contain another input of a (possibly equal) label from p, for example:

$$(\mathsf{pq}!\lambda_1;\mathsf{pq}!\lambda;\mathsf{pq}?\lambda;\mathsf{pq}?\lambda_1;\mathsf{pq}?\lambda \boxplus \mathsf{pq}!\lambda_2;\mathsf{pq}!\lambda';\mathsf{pq}?\lambda;\mathsf{pq}?\lambda_2;\mathsf{pq}?\lambda')\upharpoonright \mathsf{q} = \mathsf{p}?\lambda;(\mathsf{p}?\lambda_1;\mathsf{p}?\lambda + \mathsf{p}?\lambda_2;\mathsf{p}?\lambda')$$

In Example 6.15 we will show why we need to distinguish these two cases.
The projection of an output choice type on the other participants is defined only if it produces the same process for all branches of the choice.
The projection of an input type on the receiver is an input action followed by the projection of the rest of the type. For the other participants the projection is simply the projection of the rest of the type.
The projection of a parallel composition of pgts on a participant r is undefined if r occurs in both pgts and it is equal to $\mathbf{0}$ if r does not occur in any of them (because in this case we may take $\mathcal{G}_i$ to be any of the two pgts, since the projection of both of them on r yields $\mathbf{0}$).
We need to show that projection is well defined, i.e. that it is a partial function. The proof is easier for pgts which are bounded according to Definition 6.2, see Lemma 6.5.

We discuss now how to ensure that each player will eventually do some communication. We require that the first occurrence of each player of a pgt appears at a bounded depth in all its traces. This condition is sufficient, as shown by the proof of progress for typed networks (Theorem 6.20). To formalise it, we define the *depth of a player* p *in an sgt* G, $\mathsf{depth}(\mathsf{G},\mathsf{p})$, which uses the length function $|\,|$ of Definition 2.3, the function play given after Definition 2.5 and the new function ord given below.

Definition 6.2 (Depth)**.** *Let the two functions* $\mathsf{ord}(\tau,\mathsf{p})$ *and* $\mathsf{depth}(\mathsf{G},\mathsf{p})$ *be defined by:*

$$\mathsf{ord}(\tau,\mathsf{p}) = \begin{cases} n & \text{if } \tau = \tau_1\cdot\beta\cdot\tau_2 \text{ and } |\tau_1| = n-1 \text{ and } \mathsf{p}\notin\mathsf{play}(\tau_1) \text{ and } \mathsf{p}\in\mathsf{play}(\beta) \\ 0 & \text{otherwise} \end{cases}$$

$$\mathsf{depth}(\mathsf{G},\mathsf{p}) = \begin{cases} \sup\{\mathsf{ord}(\tau,\mathsf{p}) \mid \tau\in\mathsf{Tr}^+(\mathsf{G})\} & \text{if } \mathsf{p}\in\mathsf{play}(\mathsf{G}) \\ 0 & \text{otherwise} \end{cases}$$

We say that a pgt $\mathcal{G}$ *is* bounded *if* $\mathsf{depth}(\mathsf{G},\mathsf{p})$ *is finite for all sub-trees* G *of* $\mathcal{G}$ *and for all* p*.*

To show that $\mathcal{G}$ is bounded it is enough to check $\mathsf{depth}(\mathcal{G},\mathsf{p})$ for all $\mathsf{p}\in\mathsf{play}(\mathcal{G})$, since for any other p we have $\mathsf{depth}(\mathcal{G},\mathsf{p}) = 0$.

Note that the depth of a participant which is a player of G does not necessarily decrease in the subtrees of G. As a matter of fact, this depth can be finite in G but infinite in one of its subtrees, as shown by the following example.

Example 6.3 . *Consider* $\mathsf{G} = \mathsf{rq}!\lambda;\mathsf{rq}?\lambda;\mathsf{G}'$ *where*

$$\mathsf{G}' = \mathsf{pq}!\lambda_1;\mathsf{pq}?\lambda_1;\mathsf{rq}!\lambda_3;\mathsf{rq}?\lambda_3 \boxplus \mathsf{pq}!\lambda_2;\mathsf{pq}?\lambda_2;\mathsf{G}'$$

Then we have:

$$\mathsf{depth}(\mathsf{G},\mathsf{r}) = 1 \qquad \mathsf{depth}(\mathsf{G},\mathsf{p}) = 3 \qquad \mathsf{depth}(\mathsf{G},\mathsf{q}) = 2$$

whereas

$$\mathsf{depth}(\mathsf{G}',\mathsf{r}) = \infty \qquad \mathsf{depth}(\mathsf{G}',\mathsf{p}) = 1 \qquad \mathsf{depth}(\mathsf{G}',\mathsf{q}) = 2$$

since $\underbrace{\mathsf{pq}!\lambda_2 \cdot \mathsf{pq}?\lambda_2}_{n} \cdot \mathsf{pq}!\lambda_1 \cdot \mathsf{pq}?\lambda_1 \cdot \mathsf{rq}!\lambda_3 \in \mathsf{Tr}^+(\mathsf{G}')$ *for all* $n \geq 0$ *and* $\sup\{3+2n \mid n \geq 0\} = \infty$.

However, the depth of a participant which is a player of G but not the player of its root communications decreases in the immediate subtrees of G, as stated in the following lemma.

Lemma 6.4 . (1) *If* $\mathsf{G} = \mathsf{pq}!\boxplus_{i\in I}\lambda_i;\mathsf{G}_i$ *and* $\mathsf{r} \in \mathsf{play}(\mathsf{G})$ *and* $\mathsf{r} \neq \mathsf{p}$, *then* $\mathsf{depth}(\mathsf{G},\mathsf{r}) > \mathsf{depth}(\mathsf{G}_i,\mathsf{r})$ *for all* $i \in I$.
(2) *If* $\mathsf{G} = \mathsf{pq}?\lambda;\mathsf{G}'$ *and* $\mathsf{r} \in \mathsf{play}(\mathsf{G})$ *and* $\mathsf{r} \neq \mathsf{q}$, *then* $\mathsf{depth}(\mathsf{G},\mathsf{r}) > \mathsf{depth}(\mathsf{G}',\mathsf{r})$.

We can now show that the definition of projection given in Figure 2 is sound.

Lemma 6.5 . *If* $\mathcal{G}$ *is bounded, then* $\mathcal{G}\upharpoonright\mathsf{r}$ *is a partial function for all* r.

Proof. It is enough to consider sgts. We redefine the projection $\downarrow_{\mathsf{r}}$ as the largest relation between sgts and processes such that $(\mathsf{G},P) \in \downarrow_{\mathsf{r}}$ implies:

i) if $\mathsf{r} \notin \mathsf{play}(\mathsf{G})$, then $P = \mathbf{0}$;
ii) if $\mathsf{G} = \mathsf{rq}!\boxplus_{i\in I}\lambda_i;\mathsf{G}_i$, then $P = \bigoplus_{i\in I}\mathsf{q}!\lambda_i;P_i$ and $(\mathsf{G}_i,P_i) \in \downarrow_{\mathsf{r}}$ for all $i \in I$;
iii) if $\mathsf{G} = \mathsf{pr}!\lambda;\mathsf{G}'$, then $(\mathsf{G}',P) \in \downarrow_{\mathsf{r}}$;
iv) if $\mathsf{G} = \mathsf{pr}!\boxplus_{i\in I}\lambda_i;\mathsf{G}_i$ and $|I| > 1$, then $P = \overrightarrow{\pi};\Sigma_{i\in I}\mathsf{p}?\lambda_i;P_i$ and $(\mathsf{G}_i,\overrightarrow{\pi};\mathsf{p}?\lambda_i;P_i) \in \downarrow_{\mathsf{r}}$ for all $i \in I$;
v) if $\mathsf{G} = \mathsf{pq}!\boxplus_{i\in I}\lambda_i;\mathsf{G}_i$ and $\mathsf{r} \notin \{\mathsf{p},\mathsf{q}\}$ and $\mathsf{r} \in \mathsf{play}(\mathsf{G}_i)$, then $(\mathsf{G}_i,P) \in \downarrow_{\mathsf{r}}$ for all $i \in I$;
vi) if $\mathsf{G} = \mathsf{pr}?\lambda;\mathsf{G}'$, then $P = \mathsf{p}?\lambda;P'$ and $(\mathsf{G}',P') \in \downarrow_{\mathsf{r}}$;
vii) if $\mathsf{G} = \mathsf{pq}?\lambda;\mathsf{G}'$ and $\mathsf{r} \neq \mathsf{q}$ and $\mathsf{r} \in \mathsf{play}(\mathsf{G}')$, then $(\mathsf{G}',P) \in \downarrow_{\mathsf{r}}$.

We define equality $\mathcal{E}$ of processes to be the largest symmetric binary relation $\mathcal{R}$ on processes such that $(P,Q) \in \mathcal{R}$ implies:

(a) if $P = \bigoplus_{i\in I}\mathsf{p}!\lambda_i;P_i$, then $Q = \bigoplus_{i\in I}\mathsf{p}!\lambda_i;Q_i$ and $(P_i,Q_i) \in \mathcal{R}$ for all $i \in I$;
(b) if $P = \Sigma_{i\in I}\mathsf{p}?\lambda_i;P_i$, then $Q = \Sigma_{i\in I}\mathsf{p}?\lambda_i;Q_i$ and $(P_i,Q_i) \in \mathcal{R}$ for all $i \in I$.

It is then enough to show that the relation $\mathcal{R}_{\mathsf{r}} = \{(P,Q) \mid \exists\,\mathsf{G}\,.\,(\mathsf{G},P) \in \downarrow_{\mathsf{r}} \text{ and } (\mathsf{G},Q) \in \downarrow_{\mathsf{r}}\}$ satisfies Clauses (a) and (b) (with $\mathcal{R}$ replaced by $\mathcal{R}_{\mathsf{r}}$), since this will imply $\mathcal{R}_{\mathsf{r}} \subseteq \mathcal{E}$. Note first that $(\mathbf{0},\mathbf{0}) \in \mathcal{R}_{\mathsf{r}}$ because $(\mathsf{End},\mathbf{0}) \in \downarrow_{\mathsf{r}}$, and that $(\mathbf{0},\mathbf{0}) \in \mathcal{E}$ because Clauses (a) and (b) are vacuously satisfied by the pair $(\mathbf{0},\mathbf{0})$, which must therefore belong to $\mathcal{E}$.
The proof is by induction on $d = \mathsf{depth}(\mathsf{G},\mathsf{r})$. We only consider Clause (b), the proof for Clause (a) being similar and simpler. So, assume $(P,Q) \in \mathcal{R}_{\mathsf{r}}$ and $P = \Sigma_{i\in I}\mathsf{p}?\lambda_i;P_i$.

Case $d = 1$. In this case $\mathsf{G} = \mathsf{pr}?\lambda;\mathsf{G}'$ and $P = \mathsf{p}?\lambda;P'$ and $(\mathsf{G}',P') \in \downarrow_{\mathsf{r}}$. From $(\mathsf{G},Q) \in \downarrow_{\mathsf{r}}$ we get $Q = \mathsf{p}?\lambda;Q'$ and $(\mathsf{G}',Q') \in \downarrow_{\mathsf{r}}$. Hence Q has the required form and $(P',Q') \in \mathcal{R}_{\mathsf{r}}$.

Case $d > 1$. By definition of $\downarrow_{\mathsf{r}}$, there are five possible subcases.

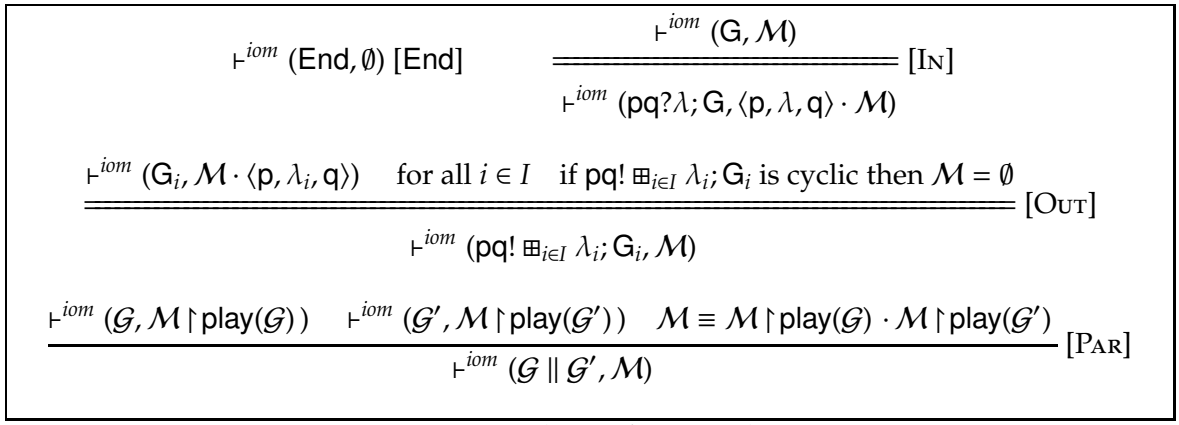

Figure 3: Input/output matching of pgts with respect to queues.

(1) Case $\mathsf{G} = \mathsf{pr}!\lambda; \mathsf{G}'$ and $(\mathsf{G}', P) \in \downarrow_{\mathsf{r}}$. From $(\mathsf{G}, Q) \in \downarrow_{\mathsf{r}}$ we get $(\mathsf{G}', Q) \in \downarrow_{\mathsf{r}}$. Then $(P, Q) \in \mathcal{R}_{\mathsf{r}}$.
(2) Case $\mathsf{G} = \mathsf{pr}! \boxplus_{i\in I} \lambda_i; \mathsf{G}_i$ and $(\mathsf{G}_i, \mathsf{p}?\lambda_i; P_i) \in \downarrow_{\mathsf{r}}$ for all $i \in I$ and $|I| > 1$. From $(\mathsf{G}, Q) \in \downarrow_{\mathsf{r}}$ we get $Q = \overrightarrow{\pi}; \sum_{i\in I} \mathsf{p}?\lambda_i; Q_i$ and $(\mathsf{G}_i, \overrightarrow{\pi}; \mathsf{p}?\lambda_i; Q_i) \in \downarrow_{\mathsf{r}}$ for all $i \in I$. Since $(\mathsf{p}?\lambda_i; P_i, \overrightarrow{\pi}; \mathsf{p}?\lambda_i; Q_i) \in \mathcal{R}_{\mathsf{r}}$ for all $i \in I$, by induction Clause (b) is satisfied. Thus $\overrightarrow{\pi} = \epsilon$ and $(P_i, Q_i) \in \mathcal{R}_{\mathsf{r}}$ for all $i \in I$.
(3) Case $\mathsf{G} = \mathsf{qr}! \boxplus_{j\in J} \lambda'_j; \mathsf{G}_j$ with $\mathsf{q} \neq \mathsf{p}$ and $P = \mathsf{p}?\lambda; \overrightarrow{\pi}; \sum_{j\in J} \mathsf{q}?\lambda'_j; P'_j$ and $(\mathsf{G}_j, \mathsf{p}?\lambda; \overrightarrow{\pi}; \mathsf{q}?\lambda'_j; P'_j) \in \downarrow_{\mathsf{r}}$ for all $j \in J$. From $(\mathsf{G}, Q) \in \downarrow_{\mathsf{r}}$ we get $Q = \overrightarrow{\pi'}; \sum_{j\in J} \mathsf{q}?\lambda'_j; Q'_j$ and $(\mathsf{G}_j, \overrightarrow{\pi'}; \mathsf{q}?\lambda'_j; Q'_j) \in \downarrow_{\mathsf{r}}$ for all $j \in J$. Since $(\mathsf{p}?\lambda; \overrightarrow{\pi}; \mathsf{q}?\lambda'_j; P'_j, \overrightarrow{\pi'}; \mathsf{q}?\lambda'_j; Q'_j) \in \mathcal{R}_{\mathsf{r}}$ for all $j \in J$, by induction Clause (b) is satisfied. Thus $\overrightarrow{\pi'} = \mathsf{p}?\lambda; \overrightarrow{\pi}$ and $(\overrightarrow{\pi}; \mathsf{q}?\lambda'_j; P'_j, \overrightarrow{\pi}; \mathsf{q}?\lambda'_j; Q'_j) \in \mathcal{R}_{\mathsf{r}}$ for all $j \in J$.
(4) Case $\mathsf{G} = \mathsf{qs}! \boxplus_{j\in J} \lambda'_j; \mathsf{G}_j$ and $\mathsf{r} \neq \mathsf{s}$ and $\mathsf{r} \in \mathsf{play}(\mathsf{G}_j)$ and $(\mathsf{G}_j, P) \in \downarrow_{\mathsf{r}}$ for $j \in J$. From $(\mathsf{G}, Q) \in \downarrow_{\mathsf{r}}$ we get $(\mathsf{G}_j, Q) \in \downarrow_{\mathsf{r}}$ for all $j \in J$. Then $(P, Q) \in \mathcal{R}_{\mathsf{r}}$.
(5) Case $\mathsf{G} = \mathsf{qs}?\lambda; \mathsf{G}'$ and $\mathsf{r} \in \mathsf{play}(\mathsf{G}')$. Then $(\mathsf{G}', P) \in \downarrow_{\mathsf{r}}$. From $(\mathsf{G}, Q) \in \downarrow_{\mathsf{r}}$ we get $(\mathsf{G}', Q) \in \downarrow_{\mathsf{r}}$. Then $(P, Q) \in \mathcal{R}_{\mathsf{r}}$. □

To ensure the correspondence between inputs and outputs, in Figure 3 we define the *input/output matching* of pgts with respect to queues. The intuition is that every input should come with a corresponding message in the queue (Rule [In]), ensuring that the input can take place. Then, each message in the queue can be exchanged for a corresponding output that will prefix the type (Rule [Out]): this output will then precede the previously inserted input and thus ensure again that the input can take place. In short, input/output matching holds if the inputs in the type are matched either by a message in the queue or by a preceding output in the type. We say that an sgt is *cyclic* if its tree contains itself as proper subtree. So the condition "if the sgt is cyclic then the queue is empty" in Rule [Out] ensures that there is no message left in the queue at the beginning of a new cycle and that all messages put in the queue by cyclic sgts have matching inputs in the same cycle. Rule [Par] requires the predicate to hold for all sgts of a parallel composition. In this rule we use the *projection of a queue on a set of participants* $\mathcal{P}$, defined as follows:

$$\emptyset \upharpoonright \mathcal{P} = \emptyset \qquad (\langle \mathsf{p}, \lambda, \mathsf{q}\rangle \cdot \mathcal{M}) \upharpoonright \mathcal{P} = \begin{cases} \langle \mathsf{p}, \lambda, \mathsf{q}\rangle \cdot (\mathcal{M} \upharpoonright \mathcal{P}) & \text{if } \mathsf{q} \in \mathcal{P} \\ \mathcal{M} \upharpoonright \mathcal{P} & \text{otherwise} \end{cases}$$

Note that if the pgt is projectable, then the queue is equivalent to the concatenation in any order of its projections on the players of the sgts which are its components.
The double line indicates that the rules are interpreted coinductively [Pie02] (Chapter 21).

The condition in Rule [Out] guarantees that we get only regular proof derivations, therefore the judgement $\vdash^{iom} (\mathcal{G}, \mathcal{M})$ is decidable.

If we derive $\vdash^{iom} (\mathcal{G}, \emptyset)$ we can ensure that in $\mathcal{G} \parallel \emptyset$ all inputs are matched by corresponding outputs and vice versa, see the Progress Theorem (Theorem 6.20). The progress property holds also for standard global types [DY11, CDCYP16].

The next example illustrates the use of the input/output predicate on a number of sgts (both cyclic and not cyclic) and queues.

Example 6.6 . (1) *The sgt* $\mathsf{qp}?\lambda;\mathsf{pq}!\lambda';\mathsf{pq}?\lambda'$ *is input/output matching for the queue* $\langle \mathsf{q}, \lambda, \mathsf{p} \rangle$*, as shown by the following derivation:*

$$\dfrac{\dfrac{\dfrac{\vdash^{iom} (\mathsf{End}, \emptyset)}{\vdash^{iom} (\mathsf{pq}?\lambda', \langle \mathsf{p}, \lambda', \mathsf{q} \rangle)}}{\vdash^{iom} (\mathsf{pq}!\lambda';\mathsf{pq}?\lambda', \emptyset)}}{\vdash^{iom} (\mathsf{qp}?\lambda;\mathsf{pq}!\lambda';\mathsf{pq}?\lambda', \langle \mathsf{q}, \lambda, \mathsf{p} \rangle)}$$

(2) *Let* $\mathsf{G} = \mathsf{pq}!\lambda;\mathsf{pq}!\lambda;\mathsf{pq}?\lambda;\mathsf{G}$*. Then* G *is not input/output matching for the empty queue. Indeed, we cannot complete the proof tree for* $\vdash^{iom} (\mathsf{G}, \emptyset)$*, since* G *is cyclic, so we cannot apply Rule* [Out] *to infer the premise* $\vdash^{iom} (\mathsf{G}, \langle \mathsf{p}, \lambda, \mathsf{q} \rangle)$ *in the following deduction:*

$$\dfrac{\dfrac{\dfrac{\vdash^{iom} (\mathsf{G}, \langle \mathsf{p}, \lambda, \mathsf{q} \rangle)}{\vdash^{iom} (\mathsf{pq}?\lambda;\mathsf{G}, \langle \mathsf{p}, \lambda, \mathsf{q} \rangle \cdot \langle \mathsf{p}, \lambda, \mathsf{q} \rangle)}}{\vdash^{iom} (\mathsf{pq}!\lambda;\mathsf{pq}?\lambda;\mathsf{G}, \langle \mathsf{p}, \lambda, \mathsf{q} \rangle)}}{\vdash^{iom} (\mathsf{G}, \emptyset)}$$

(3) *Let* $\mathsf{G}' = (\mathsf{pq}!\lambda_1;\mathsf{pq}?\lambda_1;\mathsf{G}' \boxplus \mathsf{pq}!\lambda_2;\mathsf{pq}?\lambda_2)$*. Then* G' *is input/output matching for the empty queue, as we can see from the infinite (but regular) proof tree that follows:*

$$\dfrac{\dfrac{\dfrac{\vdots}{\vdash^{iom} (\mathsf{G}', \emptyset)}}{\vdash^{iom} (\mathsf{pq}?\lambda_1;\mathsf{G}', \langle \mathsf{p}, \lambda_1, \mathsf{q} \rangle)} \qquad \dfrac{\vdash^{iom} (\mathsf{End}, \emptyset)}{\vdash^{iom} (\mathsf{pq}?\lambda_2, \langle \mathsf{p}, \lambda_2, \mathsf{q} \rangle)}}{\vdash^{iom} (\mathsf{G}', \emptyset)}$$

(4) *Let* $\mathsf{G}_1 = \mathsf{pq}!\lambda;\mathsf{pq}!\lambda;\mathsf{pq}?\lambda;\mathsf{G}_2$ *and* $\mathsf{G}_2 = \mathsf{pr}!\lambda;\mathsf{pr}?\lambda;\mathsf{G}_2$*. Then* G_1 *is not input/output matching for the empty queue. Indeed, we cannot complete the proof tree for* $\vdash^{iom} (\mathsf{G}_1, \emptyset)$*, since* G_2 *is cyclic, so we cannot apply Rule* [Out] *to infer the premise* $\vdash^{iom} (\mathsf{G}_2, \langle \mathsf{p}, \lambda, \mathsf{q} \rangle)$ *in the following deduction:*

$$\dfrac{\dfrac{\dfrac{\vdash^{iom} (\mathsf{G}_2, \langle \mathsf{p}, \lambda, \mathsf{q} \rangle)}{\vdash^{iom} (\mathsf{pq}?\lambda;\mathsf{G}_2, \langle \mathsf{p}, \lambda, \mathsf{q} \rangle \cdot \langle \mathsf{p}, \lambda, \mathsf{q} \rangle)}}{\vdash^{iom} (\mathsf{pq}!\lambda;\mathsf{pq}?\lambda;\mathsf{G}_2, \langle \mathsf{p}, \lambda, \mathsf{q} \rangle)}}{\vdash^{iom} (\mathsf{G}_1, \emptyset)}$$

Instead, G_2 *is input/output matching for the empty queue:*

$$\dfrac{\dfrac{\dfrac{\vdots}{\vdash^{iom} (\mathsf{G}_2, \emptyset)}}{\vdash^{iom} (\mathsf{pr}?\lambda; \mathsf{G}_2, \langle \mathsf{p}, \lambda, \mathsf{r} \rangle)}}{\vdash^{iom} (\mathsf{G}_2, \emptyset)}$$

Projectability, boundedness and input/output matching are the three properties that single out the agts we want to use in our type system.

Definition 6.7 (Well-formed Asynchronous Global Types)**.** *We say that the agt* $\mathcal{G} \parallel \mathcal{M}$ *is well formed if* $\mathcal{G} \upharpoonright \mathsf{p}$ *is defined for all* p*,* $\mathcal{G}$ *is bounded and* $\vdash^{iom} (\mathcal{G}, \mathcal{M})$ *is derivable.*

Clearly, it is sufficient to check that $\mathcal{G} \upharpoonright \mathsf{p}$ is defined for all $\mathsf{p} \in \mathsf{play}(\mathcal{G})$, since for any other p we have $\mathcal{G} \upharpoonright \mathsf{p} = \mathbf{0}$.

6.2. **Type System.**

We are now ready to present our type system. The unique typing rule for networks is given in Figure 4, where we assume the agt to be well formed.

We first define a preorder on processes, $P \leq Q$, meaning that *process P can be used where we expect process Q*. More precisely, $P \leq Q$ if either P is equal to Q, or we are in one of two situations: either both P and Q are output processes, sending the same labels to the same participant, and after the send P continues with a process that can be used when we expect the corresponding one in Q; or they are both input processes receiving labels from the same participant, and P may receive more labels than Q (and thus have more behaviours) but whenever it receives the same label as Q it continues with a process that can be used when we expect the corresponding one in Q. The rules are interpreted coinductively since the processes may have infinite (regular) trees.

A network $\mathsf{N} \parallel \mathcal{M}$ is typed by the agt $\mathcal{G} \parallel \mathcal{M}$ if for every participant p such that $\mathsf{p}[\![P]\!] \in \mathsf{N}$ the process P behaves as specified by the projection of $\mathcal{G}$ on p. In Rule [Net], the condition $\mathsf{play}(\mathcal{G}) \subseteq \{\mathsf{p}_i \mid i \in I\}$ ensures that all players of $\mathcal{G}$ appear in the network. Moreover it permits additional participants that do not appear in $\mathcal{G}$, allowing the typing of sessions containing $\mathsf{p}[\![\mathbf{0}]\!]$ for a fresh p — a property required to guarantee invariance of types under structural congruence of networks.

Example 6.8 . *The network of Example 2.4 can be typed by* $\mathsf{G} \parallel \emptyset$ *for four possible choices for* G*:*

$\mathsf{pq}!\lambda; \mathsf{qp}!\lambda'; \mathsf{pq}?\lambda; \mathsf{qp}?\lambda'$ $\quad$ $\mathsf{pq}!\lambda; \mathsf{qp}!\lambda'; \mathsf{qp}?\lambda'; \mathsf{pq}?\lambda$

$\mathsf{qp}!\lambda'; \mathsf{pq}!\lambda; \mathsf{pq}?\lambda; \mathsf{qp}?\lambda'$ $\quad$ $\mathsf{qp}!\lambda'; \mathsf{pq}!\lambda; \mathsf{qp}?\lambda'; \mathsf{pq}?\lambda$

$$\mathbf{0} \leq \mathbf{0}\ [\leq\text{-}\mathbf{0}] \qquad \frac{P_i \leq Q_i \quad \text{for all } i \in I}{\bigoplus_{i\in I} \mathsf{p}!\lambda_i; P_i \leq \bigoplus_{i\in I} \mathsf{p}!\lambda_i; Q_i}\ [\leq\text{-OUT}] \qquad \frac{P_i \leq Q_i \quad \text{for all } i \in I}{\sum_{i\in I\cup J} \mathsf{p}?\lambda_i; P_i \leq \sum_{i\in I} \mathsf{p}?\lambda_i; Q_i}\ [\leq\text{-IN}]$$

$$\frac{P_i \leq \mathcal{G} \upharpoonright \mathsf{p}_i \quad \text{for all } i \in I \qquad \mathsf{play}(\mathcal{G}) \subseteq \{\mathsf{p}_i \mid i \in I\}}{\vdash \prod_{i\in I} \mathsf{p}_i[\![P_i]\!] \parallel \mathcal{M} : \mathcal{G} \parallel \mathcal{M}}\ [\text{Net}]$$

Figure 4: Preorder on processes and network typing rule.

$$\mathsf{pq}!\boxplus_{i\in I}\lambda_i;\mathsf{G}_i \parallel \mathcal{M} \xrightarrow{\mathsf{pq}!\lambda_k} \mathsf{G}_k \parallel \mathcal{M}\cdot\langle \mathsf{p},\lambda_k,\mathsf{q}\rangle \text{ where } k\in I \quad [\textsc{Ext-Out}]$$

$$\mathsf{pq}?\lambda;\mathsf{G} \parallel \langle \mathsf{p},\lambda,\mathsf{q}\rangle\cdot\mathcal{M} \xrightarrow{\mathsf{pq}?\lambda} \mathsf{G} \parallel \mathcal{M} \quad [\textsc{Ext-In}]$$

$$\frac{\mathsf{G}_i \parallel \mathcal{M}\cdot\langle \mathsf{p},\lambda_i,\mathsf{q}\rangle \xrightarrow{\beta} \mathsf{G}_i' \parallel \mathcal{M}'\cdot\langle \mathsf{p},\lambda_i,\mathsf{q}\rangle \quad \text{for all } i\in I \qquad \mathsf{p}\notin \mathsf{play}(\beta)}{\mathsf{pq}!\boxplus_{i\in I}\lambda_i;\mathsf{G}_i \parallel \mathcal{M} \xrightarrow{\beta} \mathsf{pq}!\boxplus_{i\in I}\lambda_i;\mathsf{G}_i' \parallel \mathcal{M}'} \quad [\textsc{IComm-Out}]$$

$$\frac{\mathsf{G} \parallel \mathcal{M} \xrightarrow{\beta} \mathsf{G}' \parallel \mathcal{M}' \qquad \mathsf{q}\notin \mathsf{play}(\beta)}{\mathsf{pq}?\lambda;\mathsf{G} \parallel \langle \mathsf{p},\lambda,\mathsf{q}\rangle\cdot\mathcal{M} \xrightarrow{\beta} \mathsf{pq}?\lambda;\mathsf{G}' \parallel \langle \mathsf{p},\lambda,\mathsf{q}\rangle\cdot\mathcal{M}'} \quad [\textsc{IComm-In}]$$

Figure 5: LTS for simple agts.

since each participant only needs to do the output before the input. Notice that this network cannot be typed with the standard global types of [HYC16].

The network $\mathsf{N}' \parallel \mathcal{M}'$ *of Example 5.11 can be typed by the agt*

$$\mathsf{pq}?\lambda_1;\mathsf{pr}!\lambda;\mathsf{pr}?\lambda \parallel \mathcal{M}'$$

The following proposition allows us to consider only simple agts in discussing properties of the type system. The proof immediately follows by the definition of projection and the typing Rule [Net].

Proposition 6.9 . *If* $\vdash \mathsf{N} \parallel \mathcal{M} : \Pi_{i\in I}\mathsf{G}_i \parallel \mathcal{M}$, *then* $\mathsf{N} \equiv \Pi_{i\in I}\mathsf{N}_i$ *and* $\vdash \mathsf{N}_i \parallel \mathcal{M}\upharpoonright \mathsf{play}(\mathsf{G}_i) : \mathsf{G}_i \parallel \mathcal{M}\upharpoonright \mathsf{play}(\mathsf{G}_i)$ *for all* $i\in I$.

Figure 5 gives the LTS for simple agts. It shows that a communication can be performed also under a choice or an input guard, provided it has a different player.

More precisely, in Rule [IComm-Out], the premise guarantees that the communication β of G_i is independent from the choice. Notice that there are only two cases in which β could be dependent from the choice: 1) if β were another communication performed by p, which is excluded by the condition $\mathsf{p}\notin\mathsf{play}(\beta)$, and 2) if β were the matching input for the output performed by the choice. The latter situation is excluded by the requirement on the form of the queue, since in this case β would be an input by q of the last message inserted by p in the queue, but this is not possible since this message is still on the queue after the occurrence of β. Similarly, in Rule [IComm-In], the condition $\mathsf{q}\notin\mathsf{play}(\beta)$ guarantees that β is not another communication performed by the player q of the guarding input. Again, the communication β in the premise must be able to occur as if it were performed after the guarding communication, therefore it must use the queue that would result from executing the guarding communication.

We say that $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\beta} \mathsf{G}' \parallel \mathcal{M}'$ is a *top transition* if it is derived using either Rule [Ext-Out] or Rule [Ext-In]. We show that top transitions preserve the well-formedness of simple agts:

Lemma 6.10 . *If* $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\beta} \mathsf{G}' \parallel \mathcal{M}'$ *is a top transition and* $\mathsf{G} \parallel \mathcal{M}$ *is well formed, then* $\mathsf{G}' \parallel \mathcal{M}'$ *is well formed too.*

Proof. If the transition is derived using Rule [Ext-Out], then $\mathsf{G} = \mathsf{pq}! \boxplus_{i\in I} \lambda_i; \mathsf{G}_i$ and for some $k \in I$ we have $\mathsf{G}' = \mathsf{G}_k$ and $\mathcal{M}' \equiv \mathcal{M} \cdot \langle \mathsf{p}, \lambda_k, \mathsf{q}\rangle$. We show that $\mathsf{G}_k \parallel \mathcal{M} \cdot \langle \mathsf{p}, \lambda_k, \mathsf{q}\rangle$ is well formed. Since $\mathsf{G} \upharpoonright \mathsf{p}$ is defined for all p, by definition of projection also $\mathsf{G}_k \upharpoonright \mathsf{p}$ is defined for all p. Since G is bounded and G_k is a sub-tree of G, also G_k is bounded. Finally, $\vdash^{iom} (\mathsf{G}, \mathcal{M})$ implies $\vdash^{iom} (\mathsf{G}_k, \mathcal{M} \cdot \langle \mathsf{p}, \lambda_k, \mathsf{q}\rangle)$ by inversion on Rule [Out] of Figure 3.

If the transition is derived using Rule [Ext-In], then $\mathsf{G} = \mathsf{pq}?\lambda; \mathsf{G}'$ and the proof is similar and simpler. □

The following lemma detects some transitions of a simple agt from the projections of its sgt.

Lemma 6.11 . (1) *If* $\mathsf{G} \upharpoonright \mathsf{p} = \bigoplus_{i\in I} \mathsf{q}!\lambda_i; P_i$ *and* $\mathsf{G} \parallel \mathcal{M}$ *is well formed, then*

$\mathsf{G} \parallel \mathcal{M} \xrightarrow{\mathsf{pq}!\lambda_i} \mathsf{G}_i \parallel \mathcal{M} \cdot \langle \mathsf{p}, \lambda_i, \mathsf{q}\rangle$ *and* $\mathsf{G}_i \upharpoonright \mathsf{p} = P_i$ *for all* $i \in I$.

(2) *If* $\mathsf{G} \upharpoonright \mathsf{q} = \sum_{i\in I} \mathsf{p}?\lambda_i; P_i$ *and* $\mathsf{G} \parallel \mathcal{M}$ *is well formed and* $\mathcal{M} \equiv \langle \mathsf{p}, \lambda, \mathsf{q}\rangle \cdot \mathcal{M}'$ *for some* λ, *then* $I = \{k\}$ *and* $\lambda = \lambda_k$ *and* $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\mathsf{pq}?\lambda_k} \mathsf{G}' \parallel \mathcal{M}'$ *and* $\mathsf{G}' \upharpoonright \mathsf{q} = P_k$.

Proof. (1) The proof is by induction on $d = \mathsf{depth}(\mathsf{G}, \mathsf{p})$.
Case $d = 1$. By definition of projection (see Figure 2), $\mathsf{G} \upharpoonright \mathsf{p} = \bigoplus_{i\in I} \mathsf{q}!\lambda_i; P_i$ implies $\mathsf{G} = \mathsf{pq}! \boxplus_{i\in I} \lambda_i; \mathsf{G}_i$ with $\mathsf{G}_i \upharpoonright \mathsf{p} = P_i$ for all $i \in I$. Then by Rule [Ext-Out] we may conclude $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\mathsf{pq}!\lambda_i} \mathsf{G}_i \parallel \mathcal{M} \cdot \langle \mathsf{p}, \lambda_i, \mathsf{q}\rangle$ for all $i \in I$.
Case $d > 1$. In this case either i) $\mathsf{G} = \mathsf{rs}! \boxplus_{j\in J} \lambda'_j; \mathsf{G}_j$ with $\mathsf{r} \neq \mathsf{p}$ or ii) $\mathsf{G} = \mathsf{rs}?\lambda; \mathsf{G}$ with $\mathsf{s} \neq \mathsf{p}$.
i) There are three subcases.
If $\mathsf{s} = \mathsf{p}$ and $|J| = 1$, say $J = \{1\}$, then $\mathsf{G} = \mathsf{rp}!\lambda'_1; \mathsf{G}_1$. By definition of projection and by assumption $\mathsf{G} \upharpoonright \mathsf{p} = \mathsf{G}_1 \upharpoonright \mathsf{p} = \bigoplus_{i\in I} \mathsf{q}!\lambda_i; P_i$. By Lemma 6.4(1) $\mathsf{depth}(\mathsf{G}, \mathsf{p}) > \mathsf{depth}(\mathsf{G}_1, \mathsf{p})$. By Lemma 6.10 $\mathsf{G}_1 \parallel \mathcal{M} \cdot \langle \mathsf{r}, \lambda'_1, \mathsf{p}\rangle$ is well formed. Then by induction

$$\mathsf{G}_1 \parallel \mathcal{M} \cdot \langle \mathsf{r}, \lambda'_1, \mathsf{p}\rangle \xrightarrow{\mathsf{pq}!\lambda_i} \mathsf{G}'_i \parallel \mathcal{M} \cdot \langle \mathsf{r}, \lambda'_1, \mathsf{p}\rangle \cdot \langle \mathsf{p}, \lambda_i, \mathsf{q}\rangle$$

and $\mathsf{G}'_i \upharpoonright \mathsf{p} = P_i$ for all $i \in I$. Since $\mathcal{M} \cdot \langle \mathsf{r}, \lambda'_1, \mathsf{p}\rangle \cdot \langle \mathsf{p}, \lambda_i, \mathsf{q}\rangle \equiv \mathcal{M} \cdot \langle \mathsf{p}, \lambda_i, \mathsf{q}\rangle \cdot \langle \mathsf{r}, \lambda'_1, \mathsf{p}\rangle$, by Rule [IComm-Out] we get $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\mathsf{pq}!\lambda_i} \mathsf{rp}!\lambda'_1; \mathsf{G}'_i \parallel \mathcal{M} \cdot \langle \mathsf{p}, \lambda_i, \mathsf{q}\rangle$ for all $i \in I$. By definition of projection $(\mathsf{rp}!\lambda'_1; \mathsf{G}'_i) \upharpoonright \mathsf{p} = \mathsf{G}'_i \upharpoonright \mathsf{p}$ and so $(\mathsf{rp}!\lambda'_1; \mathsf{G}'_i) \upharpoonright \mathsf{p} = P_i$ for all $i \in I$.
If $\mathsf{s} = \mathsf{p}$ and $|J| > 1$, by definition of projection and the assumption that $\mathsf{G} \upharpoonright \mathsf{p}$ is a choice of output actions on q we have that $\mathsf{G} \upharpoonright \mathsf{p} = \mathsf{q}!\lambda; P$ with $P = \overrightarrow{\pi}; \sum_{j\in J} \mathsf{r}?\lambda'_j; Q_j$ and $\mathsf{G}_j \upharpoonright \mathsf{p} = \mathsf{q}!\lambda; \overrightarrow{\pi}; \mathsf{r}?\lambda'_j; Q_j$ for all $j \in J$. By Lemma 6.4(1) $\mathsf{depth}(\mathsf{G}, \mathsf{p}) > \mathsf{depth}(\mathsf{G}_j, \mathsf{p})$ for all $j \in J$. By Lemma 6.10 $\mathsf{G}_j \parallel \mathcal{M} \cdot \langle \mathsf{r}, \lambda'_j, \mathsf{s}\rangle$ is well formed. Then by induction $\mathsf{G}_j \parallel \mathcal{M} \cdot \langle \mathsf{r}, \lambda'_j, \mathsf{s}\rangle \xrightarrow{\mathsf{pq}!\lambda} \mathsf{G}'_j \parallel \mathcal{M} \cdot \langle \mathsf{r}, \lambda'_j, \mathsf{s}\rangle \cdot \langle \mathsf{p}, \lambda, \mathsf{q}\rangle$ and $\mathsf{G}'_j \upharpoonright \mathsf{p} = \overrightarrow{\pi}; \mathsf{r}?\lambda'_j; Q_j$ for all $j \in J$. Since $\mathcal{M} \cdot \langle \mathsf{r}, \lambda'_j, \mathsf{s}\rangle \cdot \langle \mathsf{p}, \lambda, \mathsf{q}\rangle \equiv \mathcal{M} \cdot \langle \mathsf{p}, \lambda, \mathsf{q}\rangle \cdot \langle \mathsf{r}, \lambda'_j, \mathsf{s}\rangle$, by Rule [IComm-Out] we get
$\mathsf{G} \parallel \mathcal{M} \xrightarrow{\mathsf{pq}!\lambda} \mathsf{rp}! \boxplus_{j\in J} \lambda'_j; \mathsf{G}'_j \parallel \mathcal{M} \cdot \langle \mathsf{p}, \lambda, \mathsf{q}\rangle$. Lastly $(\mathsf{rp}! \boxplus_{j\in J} \lambda'_j; \mathsf{G}'_j) \upharpoonright \mathsf{p} = \overrightarrow{\pi}; \sum_{j\in J} \mathsf{r}?\lambda'_j; Q_j$ since $\mathsf{G}'_j \upharpoonright \mathsf{p} = \overrightarrow{\pi}; \mathsf{r}?\lambda'_j; Q_j$. We may then conclude that $(\mathsf{rp}! \boxplus_{j\in J} \lambda'_j; \mathsf{G}'_j) \upharpoonright \mathsf{p} = P$.
If $\mathsf{s} \neq \mathsf{p}$, then by definition of projection $\mathsf{G} \upharpoonright \mathsf{p} = \mathsf{G}_j \upharpoonright \mathsf{p}$ for all $j \in J$. By Lemma 6.4(1) $\mathsf{depth}(\mathsf{G}, \mathsf{p}) > \mathsf{depth}(\mathsf{G}_j, \mathsf{p})$ for all $j \in J$. Then by induction $\mathsf{G}_j \parallel \mathcal{M} \xrightarrow{\mathsf{pq}!\lambda_i} \mathsf{G}_{i,j} \parallel \mathcal{M} \cdot \langle \mathsf{p}, \lambda_i, \mathsf{q}\rangle$ and $\mathsf{G}_{i,j} \upharpoonright \mathsf{p} = P_i$ for all $i \in I$ and all $j \in J$. By Rule [IComm-Out]

$$\mathsf{G} \parallel \mathcal{M} \xrightarrow{\mathsf{pq}!\lambda_i} \mathsf{rs}! \boxplus_{j\in J} \lambda'_j; \mathsf{G}_{i,j} \parallel \mathcal{M} \cdot \langle \mathsf{p}, \lambda_i, \mathsf{q}\rangle$$

for all $i \in I$. By definition of projection $(\mathsf{rs}! \boxplus_{j\in J} \lambda'_j; \mathsf{G}_{i,j}) \upharpoonright \mathsf{p} = \mathsf{G}_{i,j} \upharpoonright \mathsf{p} = P_i$ for all $i \in I$.
ii) The proof of this case is similar and simpler than the proof of case i). It uses Lemmas 6.4(2) and 6.10 and Rule [IComm-In], instead of Lemmas 6.4(1) and 6.10 and Rule [IComm-Out]. Note that, in order to apply Rule [IComm-In], we need $\mathcal{M} \equiv \langle \mathsf{r}, \lambda, \mathsf{s}\rangle \cdot \mathcal{M}'$. This derives from input/output matching of $\mathsf{rs}?\lambda; \mathsf{G}'$ for the queue $\mathcal{M}$ using Rule [In] of Figure 3.

(2) The proof is by induction on $d = \mathsf{depth}(\mathsf{G}, \mathsf{q})$.
Case $d = 1$. By definition of projection and the hypothesis $\mathsf{G} \upharpoonright \mathsf{q} = \Sigma_{i\in I} \mathsf{p}?\lambda_i; P_i$, it must be $\mathsf{G} = \mathsf{pq}?\lambda; \mathsf{G}'$ and $|I| = 1$, say $I = \{k\}$, and $\lambda = \lambda_k$ and $\mathsf{G}' \upharpoonright \mathsf{q} = P_k$. Then by Rule [Ext-In] we deduce $\mathsf{G} \parallel \langle \mathsf{p}, \lambda_k, \mathsf{q}\rangle \cdot \mathcal{M}' \xrightarrow{\mathsf{pq}?\lambda_k} \mathsf{G}' \parallel \mathcal{M}'$.
Case $d > 1$. In this case either i) $\mathsf{G} = \mathsf{rs}! \boxplus_{j\in J} \lambda'_j; \mathsf{G}_j$ with $\mathsf{r} \neq \mathsf{q}$ or ii) $\mathsf{G} = \mathsf{rs}?\lambda; \mathsf{G}'$ with $\mathsf{s} \neq \mathsf{q}$.
i) There are two subcases, depending on whether $\mathsf{s} = \mathsf{q}$ or $\mathsf{s} \neq \mathsf{q}$. The most interesting case is the first one, namely $\mathsf{G} = \mathsf{rq}! \boxplus_{j\in J} \lambda'_j; \mathsf{G}_j$. By definition of projection $\mathsf{G} \upharpoonright \mathsf{q} = \overrightarrow{\pi}; \Sigma_{j\in J} \mathsf{r}?\lambda'_j; Q_j$ where $\mathsf{G}_j \upharpoonright \mathsf{q} = \overrightarrow{\pi}; \mathsf{r}?\lambda'_j; Q_j$. By assumption $\mathsf{G} \upharpoonright \mathsf{q} = \Sigma_{i\in I} \mathsf{p}?\lambda_i; P_i$, thus it must be either $\overrightarrow{\pi} = \epsilon$ or $|I| = 1$, say $I = \{k\}$, and $\overrightarrow{\pi} = \mathsf{p}?\lambda_k; \overrightarrow{\pi'}$.
If $\overrightarrow{\pi} = \epsilon$, we have that $\mathsf{r} = \mathsf{p}$ and $J = I$ and $\lambda'_i = \lambda_i$ and $Q_i = P_i$ for all $i \in I$. This means that $\mathsf{G} = \mathsf{pq}! \boxplus_{i\in I} \lambda_i; \mathsf{G}_i$ and $\mathsf{G}_i \upharpoonright \mathsf{q} = \mathsf{p}?\lambda_i; P_i$. Let $\mathcal{M}_i \equiv \langle \mathsf{p}, \lambda, \mathsf{q}\rangle \cdot \mathcal{M}'_i$ where $\mathcal{M}'_i = \mathcal{M}' \cdot \langle \mathsf{p}, \lambda_i, \mathsf{q}\rangle$. By Lemma 6.10 $\mathsf{G}_i \parallel \mathcal{M}_i$ is well formed for all $i \in I$. By Lemma 6.4(1) $\mathsf{depth}(\mathsf{G}, \mathsf{q}) > \mathsf{depth}(\mathsf{G}_i, \mathsf{q})$ for all $i \in I$. By induction hypothesis, $\mathsf{G}_i \parallel \mathcal{M}_i \xrightarrow{\mathsf{pq}?\lambda} \mathsf{G}'_i \parallel \mathcal{M}'_i$ and $\lambda = \lambda_i$ and $\mathsf{G}'_i \upharpoonright \mathsf{q} = P_i$ for all $i \in I$. This implies that $|I| = 1$, say $I = \{k\}$. Then $\mathsf{G} = \mathsf{pq}!\lambda; \mathsf{G}_k$ and by Rule [IComm-Out] we deduce $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\mathsf{pq}?\lambda} \mathsf{G}' \parallel \mathcal{M}'$, where $\mathsf{G}' = \mathsf{pq}!\lambda; \mathsf{G}'_k$. Whence by definition of projection $\mathsf{G} \upharpoonright \mathsf{q} = \mathsf{G}_k \upharpoonright \mathsf{q} = \mathsf{p}?\lambda_k; P_k$ and $\mathsf{G}' \upharpoonright \mathsf{q} = \mathsf{G}'_k \upharpoonright \mathsf{q} = P_k$.
If $\overrightarrow{\pi} = \mathsf{p}?\lambda_k; \overrightarrow{\pi'}$ then $\mathsf{G} \upharpoonright \mathsf{q} = \mathsf{p}?\lambda_k; P_k$, where $P_k = \overrightarrow{\pi'}; \Sigma_{j\in J} \mathsf{r}?\lambda'_j; Q_j$. Let $\mathcal{M}_j \equiv \langle \mathsf{p}, \lambda, \mathsf{q}\rangle \cdot \mathcal{M}'_j$ where $\mathcal{M}'_j = \mathcal{M}' \cdot \langle \mathsf{r}, \lambda'_j, \mathsf{q}\rangle$. For all $j \in J$, $\mathsf{G}_j \parallel \mathcal{M}_j$ is well formed by Lemma 6.10 and $\mathsf{depth}(\mathsf{G}, \mathsf{q}) > \mathsf{depth}(\mathsf{G}_j, \mathsf{q})$ by Lemma 6.4(1). By induction hypothesis we get $\lambda = \lambda_k$ and $\mathsf{G}_j \parallel \mathcal{M}_j \xrightarrow{\mathsf{pq}?\lambda} \mathsf{G}'_j \parallel \mathcal{M}'_j$ for all $j \in J$. Let $\mathsf{G}' = \mathsf{rq}! \boxplus_{j\in J} \lambda'_j; \mathsf{G}'_j$. Then $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\mathsf{pq}?\lambda} \mathsf{G}' \parallel \mathcal{M}'$ by Rule [IComm-Out] and $\mathsf{G}' \upharpoonright \mathsf{q} = \overrightarrow{\pi'}; \Sigma_{j\in J} \mathsf{r}?\lambda'_j; Q_j = P_k$.
ii) The proof of this case is similar and simpler than the proof of case i). It uses Lemmas 6.4(2) and 6.10 and Rule [IComm-In], instead of Lemmas 6.4(1) and 6.10 and Rule [IComm-Out]. Note that, in order to apply Rule [IComm-In], we need $\mathcal{M} \equiv \langle \mathsf{r}, \lambda, \mathsf{s}\rangle \cdot \mathcal{M}'$. This derives from input/output matching of $\mathsf{rs}?\lambda; \mathsf{G}'$ for the queue $\mathcal{M}$ using Rule [In] of Figure 3. □

We can also detect the projections of an sgt from a transition of the simple agt obtained by putting the sgt in parallel with a compliant queue.

Lemma 6.12 . (1) *If* $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\mathsf{pq}!\lambda} \mathsf{G}' \parallel \mathcal{M}'$ *and* $\mathsf{G} \parallel \mathcal{M}$ *is well formed, then* $\mathcal{M}' \equiv \mathcal{M} \cdot \langle \mathsf{p}, \lambda, \mathsf{q}\rangle$ *and* $\mathsf{G} \upharpoonright \mathsf{p} = \bigoplus_{i\in I} \mathsf{q}!\lambda_i; P_i$ *and* $\lambda = \lambda_k$ *and* $\mathsf{G}' \upharpoonright \mathsf{p} = P_k$ *for some* $k \in I$ *and* $\mathsf{G} \upharpoonright \mathsf{r} \leq \mathsf{G}' \upharpoonright \mathsf{r}$ *for all* $\mathsf{r} \neq \mathsf{p}$.

(2) *If* $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\mathsf{pq}?\lambda} \mathsf{G}' \parallel \mathcal{M}'$ *and* $\mathsf{G} \parallel \mathcal{M}$ *is well formed, then* $\mathcal{M} \equiv \langle \mathsf{p}, \lambda, \mathsf{q}\rangle \cdot \mathcal{M}'$ *and* $\mathsf{G} \upharpoonright \mathsf{q} = \mathsf{pq}?\lambda; \mathsf{G}' \upharpoonright \mathsf{q}$ *and* $\mathsf{G} \upharpoonright \mathsf{r} \leq \mathsf{G}' \upharpoonright \mathsf{r}$ *for all* $\mathsf{r} \neq \mathsf{q}$.

Proof. (1) By induction on the inference of the transition $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\mathsf{pq}!\lambda} \mathsf{G}' \parallel \mathcal{M}'$.

Base Case. The applied rule must be Rule [Ext-Out], so $\mathsf{G} = \mathsf{pq}! \boxplus_{i\in I} \lambda_i; \mathsf{G}_i$ and $\lambda = \lambda_k$ and $\mathsf{G}' = \mathsf{G}_k$ for some $k \in I$, and

$$\mathsf{pq}! \boxplus_{i\in I} \lambda_i; \mathsf{G}_i \parallel \mathcal{M} \xrightarrow{\mathsf{pq}!\lambda_k} \mathsf{G}_k \parallel \mathcal{M} \cdot \langle \mathsf{p}, \lambda_k, \mathsf{q}\rangle$$

By definition of projection $\mathsf{G} \upharpoonright \mathsf{p} = \bigoplus_{i\in I} \mathsf{q}!\lambda_i; \mathsf{G}_i \upharpoonright \mathsf{p}$ and $\mathsf{G}' \upharpoonright \mathsf{p} = \mathsf{G}_k \upharpoonright \mathsf{p}$. Again by definition of projection, if $\mathsf{r} \notin \{\mathsf{p}, \mathsf{q}\}$ or $\mathsf{r} = \mathsf{q}$ and $|I| = 1$, we have $\mathsf{G} \upharpoonright \mathsf{r} = \mathsf{G}_1 \upharpoonright \mathsf{r}$ and so $\mathsf{G} \upharpoonright \mathsf{r} = \mathsf{G}' \upharpoonright \mathsf{r}$. If $\mathsf{r} = \mathsf{q}$ and $|I| > 1$, then $\mathsf{G} \upharpoonright \mathsf{q} = \overrightarrow{\pi}; \Sigma_{i\in I} \mathsf{p}?\lambda_i; Q_i$ where $\mathsf{G}_i \upharpoonright \mathsf{q} = \overrightarrow{\pi}; \mathsf{p}?\lambda_i; Q_i$ for all $i \in I$ and so $\mathsf{G} \upharpoonright \mathsf{q} \leq \mathsf{G}_k \upharpoonright \mathsf{q}$.

Inductive Cases. If the applied rule is [IComm-Out], then $\mathsf{G} = \mathsf{st}! \boxplus_{j\in J} \lambda'_j; \mathsf{G}_j$ and $\mathsf{G}' = \mathsf{st}! \boxplus_{j\in J} \lambda'_j; \mathsf{G}'_j$ and

$$\frac{\mathsf{G}_j \parallel \mathcal{M} \cdot \langle \mathsf{s}, \lambda'_j, \mathsf{t}\rangle \xrightarrow{\mathsf{pq}!\lambda} \mathsf{G}'_j \parallel \mathcal{M}' \cdot \langle \mathsf{s}, \lambda'_j, \mathsf{t}\rangle \quad j \in J \quad \mathsf{p} \neq \mathsf{s}}{\mathsf{st}! \boxplus_{j\in J} \lambda'_j; \mathsf{G}_j \parallel \mathcal{M} \xrightarrow{\mathsf{pq}!\lambda} \mathsf{st}! \boxplus_{j\in J} \lambda'_j; \mathsf{G}'_j \parallel \mathcal{M}'}$$

By Lemma 6.10 $\mathsf{G}_j \parallel \mathcal{M} \cdot \langle \mathsf{s}, \lambda'_j, \mathsf{t}\rangle$ is well formed. By induction hypothesis $\mathcal{M}' \cdot \langle \mathsf{s}, \lambda'_j, \mathsf{t}\rangle \equiv \mathcal{M} \cdot \langle \mathsf{s}, \lambda'_j, \mathsf{t}\rangle \cdot \langle \mathsf{p}, \lambda, \mathsf{q}\rangle$, which implies $\mathcal{M}' \equiv \mathcal{M} \cdot \langle \mathsf{p}, \lambda, \mathsf{q}\rangle$. If $\mathsf{p} \neq \mathsf{t}$, by definition of projection $\mathsf{G} \upharpoonright \mathsf{p} = \mathsf{G}_1 \upharpoonright \mathsf{p}$ and $\mathsf{G}_j \upharpoonright \mathsf{p} = \mathsf{G}_1 \upharpoonright \mathsf{p}$ for all $j \in J$. Similarly $\mathsf{G}' \upharpoonright \mathsf{p} = \mathsf{G}'_1 \upharpoonright \mathsf{p}$ and $\mathsf{G}'_j \upharpoonright \mathsf{p} = \mathsf{G}'_1 \upharpoonright \mathsf{p}$ for all $j \in J$. By induction hypothesis $\mathsf{G}_1 \upharpoonright \mathsf{p} = \bigoplus_{i\in I} \mathsf{q}!\lambda_i; P_i$ and $\lambda = \lambda_k$ and $\mathsf{G}'_1 \upharpoonright \mathsf{p} = P_k$ for some $k \in I$. This implies $\mathsf{G} \upharpoonright \mathsf{p} = \bigoplus_{i\in I} \mathsf{q}!\lambda_i; P_i$ and $\mathsf{G}' \upharpoonright \mathsf{p} = P_k$.

If $\mathsf{p} = \mathsf{t}$ and $|J| = 1$ the proof is as in the previous case by definition of projection.

If $\mathsf{p} = \mathsf{t}$ and $|J| > 1$, then the definition of projection gives $\mathsf{G} \upharpoonright \mathsf{p} = \overrightarrow{\pi}; \Sigma_{j\in J} \mathsf{s}?\lambda'_j; Q_j$ and $\mathsf{G}_j \upharpoonright \mathsf{p} = \overrightarrow{\pi}; \mathsf{s}?\lambda'_j; Q_j$ and $\mathsf{G}' \upharpoonright \mathsf{p} = \overrightarrow{\pi'}; \Sigma_{j\in J} \mathsf{s}?\lambda'_j; Q'_j$ and $\mathsf{G}'_j \upharpoonright \mathsf{p} = \overrightarrow{\pi'}; \mathsf{s}?\lambda'_j; Q'_j$ for all $j \in J$. By induction hypothesis $\overrightarrow{\pi} = \mathsf{q}!\lambda; \overrightarrow{\pi'}$, which implies $\mathsf{G} \upharpoonright \mathsf{p} = \mathsf{q}!\lambda; \mathsf{G}' \upharpoonright \mathsf{p}$.

For $\mathsf{r} \notin \{\mathsf{p}, \mathsf{s}, \mathsf{t}\}$ by definition of projection $\mathsf{G} \upharpoonright \mathsf{r} = \mathsf{G}_1 \upharpoonright \mathsf{r}$ and $\mathsf{G}_j \upharpoonright \mathsf{r} = \mathsf{G}_1 \upharpoonright \mathsf{r}$ for all $j \in J$. Similarly $\mathsf{G}' \upharpoonright \mathsf{r} = \mathsf{G}'_1 \upharpoonright \mathsf{r}$ and $\mathsf{G}'_j \upharpoonright \mathsf{r} = \mathsf{G}'_1 \upharpoonright \mathsf{r}$ for all $j \in J$. By induction hypothesis $\mathsf{G}_1 \upharpoonright \mathsf{r} \leq \mathsf{G}'_1 \upharpoonright \mathsf{r}$, which implies $\mathsf{G} \upharpoonright \mathsf{r} \leq \mathsf{G}' \upharpoonright \mathsf{r}$.

For participant s we have $\mathsf{G} \upharpoonright \mathsf{s} = \bigoplus_{j\in J} \mathsf{t}!\lambda'_j; \mathsf{G}_j \upharpoonright \mathsf{s} \leq \bigoplus_{j\in J} \mathsf{t}!\lambda'_j; \mathsf{G}'_j \upharpoonright \mathsf{s} = \mathsf{G}' \upharpoonright \mathsf{s}$.

For participant $\mathsf{t} \neq \mathsf{p}$ if $|J| = 1$ the proof is the same as for $\mathsf{r} \notin \{\mathsf{p}, \mathsf{s}, \mathsf{t}\}$. If $|J| > 1$, then we have $\mathsf{G} \upharpoonright \mathsf{t} = \overrightarrow{\pi}; \Sigma_{j\in J} \mathsf{s}?\lambda'_j; R_j$ where $\mathsf{G}_j \upharpoonright \mathsf{t} = \overrightarrow{\pi}; \mathsf{s}?\lambda'_j; R_j$ and $\mathsf{G}' \upharpoonright \mathsf{t} = \overrightarrow{\pi'}; \Sigma_{j\in J} \mathsf{s}?\lambda'_j; R'_j$ where $\mathsf{G}'_j \upharpoonright \mathsf{t} = \overrightarrow{\pi'}; \mathsf{s}?\lambda'_j; R'_j$. From $\mathsf{G}_j \upharpoonright \mathsf{t} \leq \mathsf{G}'_j \upharpoonright \mathsf{t}$ for all $j \in J$ we get $\overrightarrow{\pi'} = \overrightarrow{\pi}$ and $R_j \leq R'_j$ for all $j \in J$. This implies $\mathsf{G} \upharpoonright \mathsf{t} \leq \mathsf{G}' \upharpoonright \mathsf{t}$.

If the applied rule is [IComm-In] the proof is similar and simpler.

(2) The proof is similar to the proof of (1). The most interesting case is the application of Rule [IComm-Out]

$$\frac{\mathsf{G}_j \parallel \mathcal{M} \cdot \langle \mathsf{s}, \lambda'_j, \mathsf{t}\rangle \xrightarrow{\mathsf{pq}?\lambda} \mathsf{G}'_j \parallel \mathcal{M}' \cdot \langle \mathsf{s}, \lambda'_j, \mathsf{t}\rangle \quad j \in J \quad \mathsf{q} \neq \mathsf{s}}{\mathsf{st}! \boxplus_{j\in J} \lambda'_j; \mathsf{G}_j \parallel \mathcal{M} \xrightarrow{\mathsf{pq}?\lambda} \mathsf{st}! \boxplus_{j\in J} \lambda'_j; \mathsf{G}'_j \parallel \mathcal{M}'}$$

By Lemma 6.10 $\mathsf{G}_j \parallel \mathcal{M} \cdot \langle \mathsf{s}, \lambda'_j, \mathsf{t}\rangle$ is well formed. By induction hypothesis $\mathcal{M} \cdot \langle \mathsf{s}, \lambda'_j, \mathsf{t}\rangle \equiv \langle \mathsf{p}, \lambda, \mathsf{q}\rangle \cdot \mathcal{M}' \cdot \langle \mathsf{s}, \lambda'_j, \mathsf{t}\rangle$, which implies $\mathcal{M} \equiv \langle \mathsf{p}, \lambda, \mathsf{q}\rangle \cdot \mathcal{M}'$. If $\mathsf{q} \neq \mathsf{t}$, by definition of projection $\mathsf{G} \upharpoonright \mathsf{q} = \mathsf{G}_1 \upharpoonright \mathsf{q}$ and $\mathsf{G}_j \upharpoonright \mathsf{q} = \mathsf{G}_1 \upharpoonright \mathsf{q}$ for all $j \in J$. Similarly $\mathsf{G}' \upharpoonright \mathsf{q} = \mathsf{G}'_1 \upharpoonright \mathsf{q}$ and $\mathsf{G}'_j \upharpoonright \mathsf{q} = \mathsf{G}'_1 \upharpoonright \mathsf{q}$ for all $j \in J$. By induction hypothesis $\mathsf{G}_1 \upharpoonright \mathsf{q} = \mathsf{pq}?\lambda; \mathsf{G}'_1 \upharpoonright \mathsf{q}$. This implies $\mathsf{G} \upharpoonright \mathsf{q} = \mathsf{pq}?\lambda; \mathsf{G}' \upharpoonright \mathsf{p}$.

If $\mathsf{q} = \mathsf{t}$ and $|J| = 1$ the proof is as in the previous case by definition of projection.

If $\mathsf{q} = \mathsf{t}$ and $|J| > 1$, then the definition of projection gives $\mathsf{G} \upharpoonright \mathsf{q} = \overrightarrow{\pi}; \Sigma_{j\in J}\mathsf{s}?\lambda'_j; Q_j$ and $\mathsf{G}_j \upharpoonright \mathsf{q} = \overrightarrow{\pi}; \mathsf{s}?\lambda'_j; Q_j$ and $\mathsf{G}' \upharpoonright \mathsf{q} = \overrightarrow{\pi'}; \Sigma_{j\in J}\mathsf{s}?\lambda'_j; Q'_j$ and $\mathsf{G}'_j \upharpoonright \mathsf{q} = \overrightarrow{\pi'}; \mathsf{s}?\lambda'_j; Q'_j$ for all $j \in J$. By induction hypothesis $\overrightarrow{\pi} = \mathsf{p}?\lambda; \overrightarrow{\pi'}$, which implies $\mathsf{G} \upharpoonright \mathsf{q} = \mathsf{p}?\mathsf{q}\lambda; \mathsf{G}' \upharpoonright \mathsf{p}$.
The proof of $\mathsf{G} \upharpoonright \mathsf{r} \leq \mathsf{G}' \upharpoonright \mathsf{r}$ for all $\mathsf{r} \neq \mathsf{q}$ is as in case (1). □

The previous lemma will be used to show that transitions of well-formed simple agts preserve projectability of their sgt components. The LTS preserves well-formedness if also input/output matching is maintained.

Lemma 6.13 . *If* $\vdash^{iom} (\mathsf{G}, \mathcal{M})$ *and* $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\beta} \mathsf{G}' \parallel \mathcal{M}'$*, then* $\vdash^{iom} (\mathsf{G}', \mathcal{M}')$.

Proof. By induction on the inference of the transition $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\beta} \mathsf{G}' \parallel \mathcal{M}'$ of Figure 5.
Base Cases. Immediate from Lemma 6.10.
Inductive Cases. Let $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\beta} \mathsf{G}' \parallel \mathcal{M}'$ with Rule [IComm-Out]. Then we get $\mathsf{G} = \mathsf{pq}! \boxplus_{i\in I} \lambda_i; \mathsf{G}_i$ and $\mathsf{G}' = \mathsf{pq}! \boxplus_{i\in I} \lambda_i; \mathsf{G}'_i$ and $\mathsf{G}_i \parallel \mathcal{M} \cdot \langle \mathsf{p}, \lambda_i, \mathsf{q}\rangle \xrightarrow{\beta} \mathsf{G}'_i \parallel \mathcal{M}' \cdot \langle \mathsf{p}, \lambda_i, \mathsf{q}\rangle$ for all $i \in I$. From Rule [Out] of Figure 3, we get $\vdash^{iom} (\mathsf{G}_i, \mathcal{M} \cdot \langle \mathsf{p}, \lambda_i, \mathsf{q}\rangle)$ for all $i \in I$. By induction hypotheses for all $i \in I$ we can derive $\vdash^{iom} (\mathsf{G}'_i, \mathcal{M} \cdot \langle \mathsf{p}, \lambda_i, \mathsf{q}\rangle)$. Therefore using Rule [Out] we conclude $\vdash^{iom} (\mathsf{G}', \mathcal{M}')$.
Similarly for Rule [IComm-In]. □

We are now able to show that transitions preserve well-formedness of simple agts.

Lemma 6.14 . *If* $\mathsf{G} \parallel \mathcal{M}$ *is a well formed simple agt and* $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\beta} \mathsf{G}' \parallel \mathcal{M}'$*, then* $\mathsf{G}' \parallel \mathcal{M}'$ *is a well formed simple agt too.*

Proof. Let $\beta = \mathsf{pq}!\lambda$. By Lemma 6.12(1) we have that $\mathsf{G}' \upharpoonright \mathsf{r}$ is defined for all $\mathsf{r} \in \mathsf{play}(\mathsf{G})$. Similarly for $\beta = \mathsf{pq}?\lambda$, using Lemma 6.12(2). The proof that $\mathsf{depth}(\mathsf{G}'', \mathsf{r})$ is finite for all r and G'' sub-tree of G' is easy by induction on the transition rules of Figure 5.
Finally, from Lemma 6.13 we have that G' is input/output matching for the queue $\mathcal{M}'$. □

The two clauses of the projection of an output choice on the receiver, see Figure 2, are needed for the LTS to preserve projectability of well-formed simple agts, as the following example shows.

Example 6.15 . *Let* $\mathsf{G} = \mathsf{pq}!\lambda; \mathsf{pq}?\lambda; \mathsf{G}'$*, where* $\mathsf{G}' = \mathsf{qr}!\lambda_1; \mathsf{qr}?\lambda_1; \mathsf{pq}?\lambda \boxplus \mathsf{qr}!\lambda_2; \mathsf{qr}?\lambda_2; \mathsf{pq}?\lambda$*. The simple agt* $\mathsf{G} \parallel \langle \mathsf{p}, \lambda, \mathsf{q}\rangle$ *is well formed. Assume we modify the definition of projection of an output choice on the receiver by removing its first clause and the restriction of the second to* $|J| > 1$*. Then* $\mathsf{G} \upharpoonright \mathsf{q}$ *is defined since* $(\mathsf{pq}?\lambda; \mathsf{G}') \upharpoonright \mathsf{q} = \mathsf{p}?\lambda; (\mathsf{r}!\lambda_1; \mathsf{p}?\lambda \oplus \mathsf{r}!\lambda_2; \mathsf{p}?\lambda)$ *has the required shape. Applying Rule* [IComm-Out] *we get* $\mathsf{G} \parallel \langle \mathsf{p}, \lambda, \mathsf{q}\rangle \xrightarrow{\mathsf{pq}?\lambda} \mathsf{pq}!\lambda; \mathsf{G}' \parallel \emptyset$*. The projection* $(\mathsf{pq}!\lambda; \mathsf{G}') \upharpoonright \mathsf{q}$ *would not be defined since* $\mathsf{G}' \upharpoonright \mathsf{q} = \mathsf{r}!\lambda_1; \mathsf{p}?\lambda \oplus \mathsf{r}!\lambda_2; \mathsf{p}?\lambda$ *does not have the required shape.*

By virtue of Lemma 6.14, **we will henceforth only consider well-formed simple agts**.

We end this section with the expected results of Subject Reduction, Session Fidelity [HYC08, HYC16] and Progress [DY11, CDCYP16], which rely as usual on Inversion and Canonical Form lemmas.

Lemma 6.16 (Inversion)**.** *If* $\vdash \mathsf{N} \parallel \mathcal{M} : \mathsf{G} \parallel \mathcal{M}$*, then* $P \leq \mathsf{G} \upharpoonright \mathsf{p}$ *for all* $\mathsf{p}[\![P]\!] \in \mathsf{N}$.

Lemma 6.17 (Canonical Form)**.** *If* $\vdash \mathsf{N} \parallel \mathcal{M} : \mathsf{G} \parallel \mathcal{M}$ *and* $\mathsf{p} \in \mathsf{play}(\mathsf{G})$*, then* $\mathsf{p}[\![P]\!] \in \mathsf{N}$ *and* $P \leq \mathsf{G} \upharpoonright \mathsf{p}$.

Theorem 6.18 (Subject Reduction)**.** *If* $\vdash \mathsf{N} \parallel \mathcal{M} : \mathsf{G} \parallel \mathcal{M}$ *and* $\mathsf{N} \parallel \mathcal{M} \xrightarrow{\beta} \mathsf{N}' \parallel \mathcal{M}'$*, then* $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\beta} \mathsf{G}' \parallel \mathcal{M}'$ *and* $\vdash \mathsf{N}' \parallel \mathcal{M}' : \mathsf{G}' \parallel \mathcal{M}'$.

Proof. Let $\beta = \mathsf{pq}!\lambda$. By Rule [Send] of Figure 1, $\mathsf{p}[\![\bigoplus_{i\in I} \mathsf{q}!\lambda_i; P_i]\!] \in \mathsf{N}$ and $\mathsf{p}[\![P_k]\!] \in \mathsf{N}'$ and $\mathcal{M}' = \mathcal{M} \cdot \langle \mathsf{p}, \lambda_k, \mathsf{q} \rangle$ and $\lambda = \lambda_k$ for some $k \in I$. Moreover $\mathsf{r}[\![R]\!] \in \mathsf{N}$ iff $\mathsf{r}[\![R]\!] \in \mathsf{N}'$ for all $\mathsf{r} \neq \mathsf{p}$. From Lemma 6.16 we get

(1) $\bigoplus_{i\in I} \mathsf{q}!\lambda_i; P_i \leq \mathsf{G} \upharpoonright \mathsf{p}$, which implies $\mathsf{G} \upharpoonright \mathsf{p} = \bigoplus_{i\in I} \mathsf{q}!\lambda_i; P_i'$ with $P_i \leq P_i'$ for all $i \in I$ from Rule [$\leq$-out] of Figure 4 , and
(2) $R \leq \mathsf{G} \upharpoonright \mathsf{r}$ for all $\mathsf{r} \neq \mathsf{p}$ such that $\mathsf{r}[\![R]\!] \in \mathsf{N}$.

By Lemma 6.11(1) $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\mathsf{pq}\lambda_k} \mathsf{G}_k \parallel \mathcal{M} \cdot \langle \mathsf{p}, \lambda_k, \mathsf{q} \rangle$ and $\mathsf{G}_k \upharpoonright \mathsf{p} = P_k'$, which implies $P_k \leq \mathsf{G}_k \upharpoonright \mathsf{p}$. By Lemma 6.12(1) $\mathsf{G} \upharpoonright \mathsf{r} \leq \mathsf{G}_k \upharpoonright \mathsf{r}$ for all $\mathsf{r} \neq \mathsf{p}$. By transitivity of $\leq$ we have $R \leq \mathsf{G}_k \upharpoonright \mathsf{r}$ for all $\mathsf{r} \neq \mathsf{p}$. We can then choose $\mathsf{G}' = \mathsf{G}_k$.
Let $\beta = \mathsf{pq}?\lambda$. By Rule [Rcv] of Figure 1, $\mathsf{q}[\![\sum_{j\in J} \mathsf{p}?\lambda_j; Q_j]\!] \in \mathsf{N}$ and $\mathsf{q}[\![Q_k]\!] \in \mathsf{N}'$ and $\mathcal{M} = \langle \mathsf{p}, \lambda_k, \mathsf{q} \rangle \cdot \mathcal{M}'$ and $\lambda = \lambda_k$ for some $k \in J$. Moreover $\mathsf{r}[\![R]\!] \in \mathsf{N}$ iff $\mathsf{r}[\![R]\!] \in \mathsf{N}'$ for all $\mathsf{r} \neq \mathsf{q}$. From Lemma 6.16 we get

(1) $\sum_{j\in J} \mathsf{p}?\lambda_j; Q_j \leq \mathsf{G} \upharpoonright \mathsf{q}$, which implies $\mathsf{G} \upharpoonright \mathsf{q} = \sum_{j\in I} \mathsf{p}?\lambda_j; Q_j'$ with $I \subseteq J$ and $Q_i \leq Q_i'$ for all $i \in I$ from Rule [$\leq$-in] of Figure 4, and
(2) $R \leq \mathsf{G} \upharpoonright \mathsf{r}$ for all $\mathsf{r} \neq \mathsf{q}$ such that $\mathsf{r}[\![R]\!] \in \mathsf{N}$.

By Lemma 6.11(2), since $\mathcal{M} = \langle \mathsf{p}, \lambda_k, \mathsf{q} \rangle \cdot \mathcal{M}'$, we get $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\mathsf{pq}?\lambda_k} \mathsf{G}_k \parallel \mathcal{M}'$ and $I = \{k\}$ and $\mathsf{G}_k \upharpoonright \mathsf{q} = Q_k'$, which implies $Q_k \leq \mathsf{G}_k \upharpoonright \mathsf{p}$. By Lemma 6.12(2) $\mathsf{G} \upharpoonright \mathsf{r} \leq \mathsf{G}_k \upharpoonright \mathsf{r}$ for all $\mathsf{r} \neq \mathsf{q}$. By transitivity of $\leq$ we have $R \leq \mathsf{G}_k \upharpoonright \mathsf{r}$ for all $\mathsf{r} \neq \mathsf{q}$. We can then choose $\mathsf{G}' = \mathsf{G}_k$. □

Theorem 6.19 (Session Fidelity)**.** *If* $\vdash \mathsf{N} \parallel \mathcal{M} : \mathsf{G} \parallel \mathcal{M}$ *and* $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\beta} \mathsf{G}' \parallel \mathcal{M}'$*, then* $\mathsf{N} \parallel \mathcal{M} \xrightarrow{\beta} \mathsf{N}' \parallel \mathcal{M}'$ *and* $\vdash \mathsf{N}' \parallel \mathcal{M}' : \mathsf{G}' \parallel \mathcal{M}'$.

Proof. Let $\beta = \mathsf{pq}!\lambda$. By Lemma 6.12(1) $\mathcal{M}' \equiv \mathcal{M} \cdot \langle \mathsf{p}, \lambda, \mathsf{q} \rangle$, $\mathsf{G} \upharpoonright \mathsf{p} = \bigoplus_{i\in I} \mathsf{p}!\lambda_i; P_i$, $\lambda = \lambda_k$, $\mathsf{G}' \upharpoonright \mathsf{p} = P_k$ for some $k \in I$ and $\mathsf{G} \upharpoonright \mathsf{r} \leq \mathsf{G}' \upharpoonright \mathsf{r}$ for all $\mathsf{r} \neq \mathsf{p}$. From Lemma 6.17 we get $\mathsf{N} \equiv \mathsf{p}[\![P]\!] \parallel \mathsf{N}''$ and

(1) $P = \bigoplus_{i\in I} \mathsf{q}!\lambda_i; P_i'$ with $P_i' \leq P_i$ for all $i \in I$, from Rule [$\leq$-out] of Figure 4, and
(2) $R \leq \mathsf{G} \upharpoonright \mathsf{r}$ for all $\mathsf{r}[\![R]\!] \in \mathsf{N}''$.

We can then choose $\mathsf{N}' = \mathsf{p}[\![P_k']\!] \parallel \mathsf{N}''$.
Let $\beta = \mathsf{pq}?\lambda$. By Lemma 6.12(2) $\mathcal{M} \equiv \langle \mathsf{p}, \lambda, \mathsf{q} \rangle \cdot \mathcal{M}'$, $\mathsf{G} \upharpoonright \mathsf{q} = \mathsf{p}?\lambda; P$, $\mathsf{G}' \upharpoonright \mathsf{q} = P$ and $\mathsf{G} \upharpoonright \mathsf{r} \leq \mathsf{G}' \upharpoonright \mathsf{r}$ for all $\mathsf{r} \neq \mathsf{q}$. From Lemma 6.17 we get $\mathsf{N} \equiv \mathsf{q}[\![Q]\!] \parallel \mathsf{N}''$ and

(1) $Q = \mathsf{p}?\lambda; P' + Q'$ with $P' \leq P$, from Rule [$\leq$-in] of Figure 4, and
(2) $R \leq \mathsf{G} \upharpoonright \mathsf{r}$ for all $\mathsf{r}[\![R]\!] \in \mathsf{N}''$.

We can then choose $\mathsf{N}' = \mathsf{q}[\![P']\!] \parallel \mathsf{N}''$. □

We are now able to prove that in a typable network, every participant whose process is not terminated may eventually perform an action, and every message that is stored in the queue is eventually received. This property is generally referred to as progress.

Theorem 6.20 (Progress)**.** *If* $\vdash \mathsf{N}_0 \parallel \mathcal{M}_0 : \mathsf{G}_0 \parallel \mathcal{M}_0$*, then the network* $\mathsf{N}_0 \parallel \mathcal{M}_0$ *satisfies progress, namely, if either* $\mathsf{N} \parallel \mathcal{M} = \mathsf{N}_0 \parallel \mathcal{M}_0$ *or* $\mathsf{N}_0 \parallel \mathcal{M}_0 \xrightarrow{\tau} \mathsf{N} \parallel \mathcal{M}$*, then:*

(1) $\mathsf{p}[\![\, P \,]\!] \in \mathsf{N}$ *and* $P \neq \mathbf{0}$ *imply* $\mathsf{N} \parallel \mathcal{M} \xrightarrow{\tau' \cdot \beta} \mathsf{N}' \parallel \mathcal{M}'$ *with* $\mathsf{play}(\beta) = \{\mathsf{p}\}$;

(2) $\mathcal{M} \equiv \langle \mathsf{p}, \lambda, \mathsf{q} \rangle \cdot \mathcal{M}_1$ *implies* $\mathsf{N} \parallel \mathcal{M} \xrightarrow{\tau' \cdot \mathsf{pq}?\lambda} \mathsf{N}' \parallel \mathcal{M}'$.

Proof. (1) If P is an output process, then it can always move. If P is an input process, then by Subject Reduction (Theorem 6.18) $\vdash \mathsf{N} \parallel \mathcal{M} : \mathsf{G} \parallel \mathcal{M}$ for some G. We prove by induction on $d = \mathsf{depth}(\mathsf{G}, \mathsf{p})$ that $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\tau' \cdot \beta} \mathsf{G}' \parallel \mathcal{M}'$ with $\mathsf{play}(\beta) = \{\mathsf{p}\}$. This will imply $\mathsf{N} \parallel \mathcal{M} \xrightarrow{\tau' \cdot \beta} \mathsf{N}' \parallel \mathcal{M}'$ by Session Fidelity (Theorem 6.19).
Case $d = 1$. Here $\mathsf{G} = \mathsf{qp}?\lambda; \mathsf{G}'$. Since $\mathsf{G} \parallel \mathcal{M}$ is well formed, this implies $\mathcal{M} \equiv \langle \mathsf{q}, \lambda, \mathsf{q} \rangle \cdot \mathcal{M}'$ by Rule [In] of Figure 3. Then $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\mathsf{qp}?\lambda} \mathsf{G}' \parallel \mathcal{M}'$ by Rule [Ext-In] of Figure 5.
Case $d > 1$. Here we have either $\mathsf{G} = \mathsf{rs}! \boxplus_{i \in I} \lambda_i; \mathsf{G}_i$ with $\mathsf{r} \neq \mathsf{p}$ or $\mathsf{G} = \mathsf{rs}?\lambda; \mathsf{G}'''$ with $\mathsf{s} \neq \mathsf{p}$. By Lemma 6.4 this implies $\mathsf{depth}(\mathsf{G}_i, \mathsf{p}) < d$ for all $i \in I$ in the first case, and $\mathsf{depth}(\mathsf{G}''', \mathsf{p}) < d$ in the second case. Hence in both cases, by applying Rule [Ext-Out] or Rule [Ext-In] of Figure 5, we get $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\beta'} \mathsf{G}'' \parallel \mathcal{M}''$ with $\mathsf{depth}(\mathsf{G}'', \mathsf{p}) < d$. By induction $\mathsf{G}'' \parallel \mathcal{M}'' \xrightarrow{\tau' \cdot \beta} \mathsf{G}' \parallel \mathcal{M}'$ with $\mathsf{play}(\beta) = \{\mathsf{p}\}$. Therefore $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\beta' \cdot \tau' \cdot \beta} \mathsf{G}' \parallel \mathcal{M}'$ is the required transition sequence.

(2) We define the *input depth* of the input $\mathsf{pq}?\lambda$ in G, notation $\mathsf{idepth}(\mathsf{G}, \mathsf{pq}?\lambda)$, by induction on G:

$$\begin{array}{lcl} \mathsf{idepth}(\mathsf{rs}! \boxplus_{i \in I} \lambda_i; \mathsf{G}_i, \mathsf{pq}?\lambda) & = & 1 + max_{i \in I} \mathsf{idepth}(\mathsf{G}_i, \mathsf{pq}?\lambda) \\ \mathsf{idepth}(\mathsf{rs}?\lambda'; \mathsf{G}', \mathsf{pq}?\lambda) & = & \begin{cases} 1 & \text{if } \mathsf{pq}?\lambda = \mathsf{rs}?\lambda' \\ 1 + \mathsf{idepth}(\mathsf{G}', \mathsf{pq}?\lambda) & \text{otherwise} \end{cases} \end{array}$$

Notice that $\vdash^{iom} (\mathsf{G}, \langle \mathsf{p}, \lambda, \mathsf{q} \rangle \cdot \mathcal{M}_1)$ implies that $\mathsf{idepth}(\mathsf{G}, \mathsf{pq}?\lambda)$ is finite, since proof derivations are regular and only Rule [In] of Figure 3 adds messages to the queue.
By Subject Reduction (Theorem 6.18) $\vdash \mathsf{N} \parallel \mathcal{M} : \mathsf{G} \parallel \mathcal{M}$ for some G. We prove by induction on $id = \mathsf{idepth}(\mathsf{G}, \mathsf{p})$ that $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\tau' \cdot \mathsf{pq}?\lambda} \mathsf{G}' \parallel \mathcal{M}'$. This will imply $\mathsf{N} \parallel \mathcal{M} \xrightarrow{\tau' \cdot \mathsf{pq}?\lambda} \mathsf{N}' \parallel \mathcal{M}'$ by Session Fidelity (Theorem 6.19).

Case $id = 1$. Here $\mathsf{G} = \mathsf{pq}?\lambda; \mathsf{G}'$, which implies $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\mathsf{pq}?\lambda} \mathsf{G}' \parallel \mathcal{M}_1$ by Rule [Ext-In] of Figure 5.
Case $id > 1$. By applying Rule [Ext-Out] or Rule [Ext-In] of Figure 5 we get

$$\mathsf{G} \parallel \mathcal{M} \xrightarrow{\beta} \mathsf{G}'' \parallel \mathcal{M}''$$

and $\mathsf{idepth}(\mathsf{G}'', \mathsf{pq}?\lambda) < id$. By induction $\mathsf{G}'' \parallel \mathcal{M}'' \xrightarrow{\tau' \cdot \mathsf{pq}?\lambda} \mathsf{G}' \parallel \mathcal{M}'$. We conclude $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\beta \cdot \tau' \cdot \mathsf{pq}?\lambda} \mathsf{G}' \parallel \mathcal{M}'$. □

The iteration of Theorem 6.20(2) ensures that each message in the queue is eventually read. The input/output matching condition is needed to avoid orphan messages, i.e. messages which remain forever in the queue, as shown by Example 6.6(4). The boundedness condition guarantees that each player of an sgt G occurs as player in every path starting from the root of G. This fails for player r in the sgt of Example 6.3.

The proof of Theorem 6.20 shows that the desired transition sequences use only Rules [Ext-Out] and [Ext-In] and the output choice is arbitrary. Moreover the lengths of these transition sequences are bounded by $\mathsf{depth}(\mathsf{G}, \mathsf{p})$ and $\mathsf{idepth}(\mathsf{G}, \mathsf{pq}?\lambda)$, respectively.

7. Event Structure Semantics of Asynchronous Global Types

We define now the event structure associated with a simple agt. The events of this ES will be equivalence classes of pairs whose elements are particular traces. The first elements are o-traces corresponding to the messages in the queue, as in n-events (Definition 5.3(1)). The second elements are traces in sgt trees.

For traces τ, as given in Definition 2.3, we use the following notational conventions:

– We denote by $\tau[i]$ the i-th element of τ, $i > 0$.
– If $i \leq j$, we define $\tau[i \,..\, j] = \tau[i] \cdots \tau[j]$ to be the subtrace of τ consisting of the $(j - i + 1)$ elements starting from the i-th one and ending with the j-th one. If $i > j$, we define $\tau[i \,..\, j]$ to be the empty trace ϵ.

If not otherwise stated we assume that τ has n elements, so $\tau = \tau[1 \,..\, n]$.

In the traces we want to require that every input matches a corresponding output. This is checked using the *multiplicity of* $\mathsf{pq}\dagger$ *in* τ, defined by induction as follows:

$$\mathsf{m}(\mathsf{pq}\dagger, \epsilon) = 0 \qquad \mathsf{m}(\mathsf{pq}\dagger, \beta \cdot \tau) = \begin{cases} \mathsf{m}(\mathsf{pq}\dagger, \tau) + 1 & \text{if } \beta = \mathsf{pq}\dagger\lambda \\ \mathsf{m}(\mathsf{pq}\dagger, \tau) & \text{otherwise} \end{cases}$$

where $\dagger \in \{!, ?\}$ (as in Definition 5.4).
An input of q from p matches a preceding output from p to q in a trace if it has the same label λ and the number of inputs from p to q in the subtrace before the given input is equal to the number of outputs from p to q in the subtrace before the given output.
This is formalised using the above multiplicity and the positions of communications in traces.

Definition 7.1 (Matching)**.** *The input* $\tau[j] = \mathsf{pq}?\lambda$ matches *the output* $\tau[i] = \mathsf{pq}!\lambda$ *in* τ*, dubbed* $i \propto^{\tau} j$*, if* $i < j$ *and* $\mathsf{m}(\mathsf{pq}!, \tau[1 \,..\, i-1]) = \mathsf{m}(\mathsf{pq}?, \tau[1 \,..\, j-1])$.

For example, if $\tau = \mathsf{pq}!\lambda; \mathsf{pq}!\lambda; \mathsf{pq}!\lambda; \mathsf{pq}?\lambda; \mathsf{pq}?\lambda$, then $1 \propto^{\tau} 4$ and $2 \propto^{\tau} 5$, while no input matches the output at position 3.

As mentioned earlier, o-traces will be used to represent queues and general traces are paths in sgt trees. We want to define an equivalence relation on general traces, which allows us to exchange the order of adjacent communications when this order is not essential. This is the case if the communications have different players and in addition they are not matching according to Definition 7.1. However, the matching relation must also take into account the fact that some outputs are already on the queue. So we will consider well-formedness with respect to a prefixing o-trace. We proceed as follows:

– we start with well-formed traces (Definition 7.2);
– we define the swapping relation $\triangleright_{\omega}$ which allows two communications to be interchanged in a trace τ, when these communications are independent in the trace $\omega \cdot \tau$ (Definition 7.3);
– then we show that $\triangleright_{\omega}$ preserves ω-well-formedness (Lemma 7.4);
– finally we define the equivalence $\approx_{\omega}$ on ω-well-formed traces (Definition 7.5).

In a well-formed trace each input must have a corresponding output. We also need a notion of well-formedness for a suffix of a trace w.r.t. the whole trace.

Definition 7.2 (Well-formedness)**.**

(1) *A trace* τ *is* well formed *if every input matches an output in* τ*.*
(2) *A trace* τ *is* τ'-well formed *if* $\tau' \cdot \tau$ *is well formed.*

As an example, the trace $\tau = \mathsf{pq}!\lambda \cdot \mathsf{pq}!\lambda' \cdot \mathsf{pq}?\lambda'$ is not well formed since $2 \not\propto^{\tau} 3$. On the other hand, τ is $\mathsf{pq}!\lambda'$-well formed, since $\mathsf{pq}!\lambda' \cdot \tau$ is well formed given that $1 \propto^{\mathsf{pq}!\lambda' \cdot \tau} 4$.

Notice that any o-trace is well formed and any well-formed trace of length 1 must be an output. A well-formed trace of length 2 can consist of either two outputs or an output followed by the matching input.

Definition 7.3 (Swapping)**.** *Let τ be ω-well formed. We say that τ ω-*swaps *to τ', notation $\tau \triangleright_\omega \tau'$, if*

$$\tau = \tau[1 \dots i-1] \cdot \beta \cdot \beta' \cdot \tau'' \qquad \tau' = \tau[1 \dots i-1] \cdot \beta' \cdot \beta \cdot \tau'' \quad \textit{and}$$
$$\mathsf{play}(\beta) \cap \mathsf{play}(\beta') = \emptyset \quad \textit{and} \quad \neg(i + |\omega| \propto^{\omega \cdot \tau} i + 1 + |\omega|)$$

For instance, if $\omega = \mathsf{pq}!\lambda$ and $\tau = \mathsf{pq}!\lambda \cdot \mathsf{pq}?\lambda$, then τ ω-swaps to $\tau' = \mathsf{pq}?\lambda \cdot \mathsf{pq}!\lambda$ because the input in τ matches the output in ω.

Lemma 7.4 . *If τ is ω-well formed and $\tau \triangleright_\omega \tau'$, then τ' is ω-well formed too.*

Proof. Let $\tau = \tau[1 \dots i-1] \cdot \beta \cdot \beta' \cdot \tau_1$ and $\tau' = \tau[1 \dots i-1] \cdot \beta' \cdot \beta \cdot \tau_1$. We want to prove that $\omega \cdot \tau'$ is well formed. To this end, we will show that if β or β' is an input, then it matches an output that occurs in the prefix $(\omega \cdot \tau)[1 \dots i-1+|\omega|]$ of $\omega \cdot \tau'$. Note that it must be $\beta \neq \beta'$, since by hypothesis $\mathsf{play}(\beta) \cap \mathsf{play}(\beta') = \emptyset$.

Suppose β' is an input. Since τ is ω-well formed, β' matches an output in $(\omega \cdot \tau)[1 \dots i-1+|\omega|] \cdot \beta$. This output cannot be β, since by hypothesis $\neg(i + |\omega| \propto^{\omega \cdot \tau} i + 1 + |\omega|)$. Hence β' matches an output which occurs in the prefix $(\omega \cdot \tau)[1 \dots i-1+|\omega|]$ of $\omega \cdot \tau'$.

Suppose now $\beta = \mathsf{pq}?\lambda$ and $\mathsf{m}(\mathsf{pq}?, (\omega \cdot \tau)[1 \dots i-1+|\omega|]) = m$. Since τ is ω-well formed, β matches an output $(\omega \cdot \tau)[j] = \mathsf{pq}!\lambda$ in the prefix $(\omega \cdot \tau)[1 \dots i-1+|\omega|]$ of $\omega \cdot \tau$. Then $1 \leq j < i+|\omega|$ and $\mathsf{m}(\mathsf{pq}!, (\omega \cdot \tau)[1 \dots j-1+|\omega|]) = m$. Since $\beta \neq \beta'$, also $\mathsf{m}(\mathsf{pq}?, (\omega \cdot \tau')[1 \dots i+|\omega|]) = m$. Then β matches $(\omega \cdot \tau)[j]$ also in $\omega \cdot \tau'$. □

From the previous lemma and the observation that if τ is ω-well formed and τ' is obtained by swapping the i-th and $(i+1)$-th element of τ, then $\mathsf{play}(\tau'[i]) \cap \mathsf{play}(\tau'[i+1]) = \emptyset$ and $\neg(i + |\omega| \propto^{\omega \cdot \tau'} i + 1 + |\omega|)$, we deduce that the swapping relation is symmetric. This allows us to define $\approx_\omega$ as the equivalence relation induced by the swapping relation.

Definition 7.5 (Equivalence $\approx_\omega$ on ω-well-formed traces)**.** *The equivalence $\approx_\omega$ on ω-well-formed traces is the reflexive and transitive closure of $\triangleright_\omega$.*

Observe that for o-traces all the equivalences $\approx_\omega$ collapse to $\approx_\epsilon$ and $\approx_\epsilon \subset \cong$, where $\cong$ is the o-trace equivalence given in Definition 5.2. Indeed, it should be clear that $\approx_\epsilon \subseteq \cong$. To show $\approx_\epsilon \neq \cong$, consider $\tau = \mathsf{pq}!\lambda \cdot \mathsf{pr}!\lambda'$ and $\tau' = \mathsf{pr}!\lambda' \cdot \mathsf{pq}!\lambda$. Then $\tau \cong \tau'$ but $\tau \not\approx_\epsilon \tau'$. This agrees with the fact that o-traces represent messages in queues, while general traces represent future communication actions.

Another constraint that we want to impose on traces in order to build events is that each communication must be a cause of at least one of those that follow it. This happens when:

– either the two communications have the same player, in which case we say that the first communication is required in the trace (Definition 7.6);
– or the first communication is an output and the second is the matching input.

We call pointedness the property of a trace in which each communication, except the last one, satisfies one of the two conditions above. Like well-formedness, also pointedness is parameterised on traces. We first define required communications.

Definition 7.6 (Required communication)**.** *We say that* $\tau[i]$ *is* required in τ, *notation* $\mathsf{req}(i, \tau)$, *if* $\mathsf{play}(\tau[i]) \subseteq \mathsf{play}(\tau[(i+1) \dots n])$, *where* $n = |\tau|$.

Note that by definition the last element $\tau[n]$ is not required in τ.

Definition 7.7 (Pointedness)**.** *The trace* τ *is* τ'-pointed *if* τ *is* τ'*-well formed and for all* i, $1 \leq i < n$, *one of the following holds:*
(1) *either* $\mathsf{req}(i, \tau)$
(2) *or* $i + |\tau'| \propto^{\tau' \cdot \tau} j + |\tau'|$ *for some* $j > i$.

Observe that the two conditions of the above definition are reminiscent of the two kinds of causality - local flow and cross-flow - discussed for network events in Section 5 (Definition 5.6). Indeed, Condition (1) holds if $\tau[i]$ is a local cause of some $\tau[j]$, $j > i$, while Condition (2) holds if $\tau[i]$ is a cross-cause of some $\tau[j]$, $j > i$.

Note also that the conditions of Definition 7.7 must be satisfied only by every $\tau[i]$ with $i < n$, thus they hold vacuously for any single communication and for the empty trace. If $\tau = \tau_1 \cdot \beta \cdot \beta'$ is τ'-pointed, then either $\mathsf{play}(\beta) = \mathsf{play}(\beta')$ or β' matches β in $\tau' \cdot \tau$. Also, if a trace τ is τ'-pointed for some τ', we know that each communication in τ must be executed before the last one.

Example 7.8 . *Let* $\omega = \mathsf{pq}!\lambda \cdot \mathsf{rq}!\lambda$ *and* $\tau = \mathsf{pq}!\lambda \cdot \mathsf{pq}?\lambda \cdot \mathsf{rq}?\lambda$. *The trace* τ *is not* ω*-pointed, since the output* $\mathsf{pq}!\lambda$ *in* τ *is not matched by any input in* $\omega \cdot \tau$ *(the input* $\mathsf{pq}?\lambda$ *in* τ *matches the output* $\mathsf{pq}!\lambda$ *in* ω*) and it is not required in* τ *because its player* p *is neither a player of* $\mathsf{pq}?\lambda$ *nor a player of* $\mathsf{rq}?\lambda$. *So the condition of Definition 7.7 is not satisfied for the output* $\mathsf{pq}!\lambda$ *in* τ. *Instead the trace* $\tau' = \mathsf{pq}?\lambda \cdot \mathsf{rq}?\lambda$ *is* ω*-pointed, as well as the trace* $\tau'' = \mathsf{rq}?\lambda \cdot \mathsf{pq}?\lambda$.

Pointedness is preserved by suffixing.

Lemma 7.9 . *If* τ *is* τ'*-pointed and* $\tau = \tau_1 \cdot \tau_2$, *then* τ_2 *is* $\tau' \cdot \tau_1$*-pointed.*

Proof. Immediate, since $(\tau' \cdot \tau_1) \cdot \tau_2 = \tau' \cdot (\tau_1 \cdot \tau_2)$ and τ_2 is a suffix of τ and therefore its elements are a subset of those of τ. □

Note on the other hand that if τ is τ'-pointed and $\tau' = \tau'_1 \cdot \tau'_2$, then it is not true that $\tau'_2 \cdot \tau$ is τ'_1-pointed, because in this case the set of elements of $\tau'_2 \cdot \tau$ is a superset of that of τ. For instance, if $\tau'_1 = \epsilon$, $\tau'_2 = \mathsf{pq}!\lambda$ and $\tau = \mathsf{rs}!\lambda' \cdot \mathsf{rs}?\lambda'$, then $\tau'_2 \cdot \tau$ is not τ'_1-pointed.

A useful property of ω-pointedness is that it is preserved by the equivalence $\approx_\omega$, which does not change the rightmost communication in ω-pointed traces. We use $\mathsf{last}(\tau)$ to denote the last communication of τ.

Lemma 7.10 . *Let* τ *be* ω*-pointed and* $\tau \approx_\omega \tau'$. *Then* τ' *is* ω*-pointed and* $\mathsf{last}(\tau') = \mathsf{last}(\tau)$.

Proof. Let $\tau \approx_\omega \tau'$. By Definition 7.5 τ' is obtained from τ by m swaps of adjacent communications. The proof is by induction on the number m of swaps.
Case $m = 0$. The result is obvious.
Case $m > 0$. In this case there is τ_1 obtained from τ by $m - 1$ swaps of adjacent communications and there are β, β', τ_2 such that

$$\tau_1 = \tau_1[1 \dots i-1] \cdot \beta \cdot \beta' \cdot \tau_2 \approx_\omega \tau_1[1 \dots i-1] \cdot \beta' \cdot \beta \cdot \tau_2 = \tau'$$
$$\text{and } \mathsf{play}(\beta) \cap \mathsf{play}(\beta') = \emptyset \text{ and } \neg(i + |\omega| \propto^{\omega \cdot \tau_1} i + 1 + |\omega|)$$

By induction hypothesis τ_1 is ω-pointed and $\mathsf{last}(\tau_1) = \mathsf{last}(\tau)$.

To show that τ' is ω-pointed, observe that $\mathsf{play}(\beta) \cap \mathsf{play}(\beta') = \emptyset$ implies:

$$\mathsf{play}(\beta) \subseteq \mathsf{play}(\beta') \cup \mathsf{play}(\tau_2) \;\Leftrightarrow\; \mathsf{play}(\beta) \subseteq \mathsf{play}(\tau_2)$$
$$\mathsf{play}(\beta') \subseteq \mathsf{play}(\tau_2) \;\Leftrightarrow\; \mathsf{play}(\beta') \subseteq \mathsf{play}(\beta) \cup \mathsf{play}(\tau_2)$$

From this we deduce $\mathsf{req}(i, \tau_1) \iff \mathsf{req}(i, \tau')$ and $\mathsf{req}(i+1, \tau_1) \iff \mathsf{req}(i+1, \tau')$, so if both $\tau_1[i]$ and $\tau_1[i+1]$ are required in τ_1 we are done.

Otherwise, suppose that $i + |\omega| \propto^{\omega \cdot \tau_1} j + |\omega|$ where either $\mathsf{req}(j, \tau_1)$ or $j = n$. If $\mathsf{req}(j, \tau_1)$ then also $\mathsf{req}(j, \tau')$, as we just saw. Now, j cannot be $i+1$ since by hypothesis $\neg(i+|\omega| \propto^{\omega \cdot \tau_1} i+1+|\omega|)$. This implies $i + 1 + |\omega| \propto^{\omega \cdot \tau'} j + |\omega|$. Similarly we can show that $i + 1 + |\omega| \propto^{\omega \cdot \tau_1} j + |\omega|$ implies $i + |\omega| \propto^{\omega \cdot \tau'} j + |\omega|$. Therefore τ' is ω-pointed.

To show that $\mathsf{last}(\tau) = \mathsf{last}(\tau')$, assume ad absurdum that $\tau_2 = \epsilon$. Then $\tau_1[1 \dots i-1] \cdot \beta \cdot \beta'$ is ω-pointed and thus, as observed after Definition 7.7, we have either $\mathsf{play}(\beta) \cap \mathsf{play}(\beta') \neq \emptyset$ or $i + |\omega| \propto^{\omega \cdot \tau_1} i+1+|\omega|$. In both cases β and β' cannot be swapped. So it must be $\tau_2 \neq \epsilon$. □

We now relate simple asynchronous global types with pairs of o-traces and traces.

Lemma 7.11 . *If $\mathsf{G} \parallel \mathcal{M}$ is a simple agt and $\omega = \mathsf{otr}(\mathcal{M})$ and $\tau \in \mathsf{Tr}^+(\mathsf{G})$, then τ is ω-well formed.*

Proof. We prove by induction on τ that $\vdash^{iom} (\mathsf{G}, \mathcal{M})$ implies that $\mathsf{otr}(\mathcal{M}) \cdot \tau$ is well formed.
Case $\tau = \beta$. If β is an output the result is obvious. If $\beta = \mathsf{pq}?\lambda$, by Rule [In] of Figure 3, we get $\mathcal{M} \equiv \langle \mathsf{p}, \lambda, \mathsf{q} \rangle \cdot \mathcal{M}'$. Therefore $\omega = \mathsf{pq}!\lambda \cdot \omega'$ and $\omega \cdot \beta$ is well formed.
Case $\tau = \beta \cdot \tau'$ *with* $\tau' \in \mathsf{Tr}^+(\mathsf{G}')$. If $\beta = \mathsf{pq}!\lambda$, then $\mathsf{G} = \mathsf{pq}! \boxplus_{i \in I} \lambda_i; \mathsf{G}_i$ and $\lambda = \lambda_k$ and $\mathsf{G}' = \mathsf{G}_k$ for some $k \in I$. From $\vdash^{iom} (\mathsf{G}, \mathcal{M})$ and Rule [Out] of Figure 3, we get $\vdash^{iom} (\mathsf{G}', \mathcal{M} \cdot \langle \mathsf{p}, \lambda, \mathsf{q} \rangle)$. By induction hypothesis on τ', $\mathsf{otr}(\mathcal{M} \cdot \langle \mathsf{p}, \lambda, \mathsf{q} \rangle) \cdot \tau'$ is well formed. So since $\mathsf{otr}(\mathcal{M} \cdot \langle \mathsf{p}, \lambda, \mathsf{q} \rangle) = \omega \cdot \mathsf{pq}!\lambda$ we get that $\omega \cdot \tau$ is well formed.
If $\beta = \mathsf{pq}?\lambda$, then $\mathsf{G} = \mathsf{pq}?\lambda; \mathsf{G}'$. From $\vdash^{iom} (\mathsf{G}, \mathcal{M})$ and Rule [In] of Figure 3, we get $\mathcal{M} \equiv \langle \mathsf{p}, \lambda, \mathsf{q} \rangle \cdot \mathcal{M}'$ and $\vdash^{iom} (\mathsf{G}', \mathcal{M}')$. Let $\omega' = \mathsf{otr}(\mathcal{M}')$. Then $\omega \cong \mathsf{pq}!\lambda \cdot \omega'$. By induction hypothesis on τ' the trace $\omega' \cdot \tau'$ is well formed. We want now to show that also the trace $\tau'' = \omega \cdot \tau = \mathsf{pq}!\lambda \cdot \omega' \cdot \mathsf{pq}?\lambda \cdot \tau'$ is well formed, namely that in τ'' every input matches an output. Note that the first input in τ'' is $\tau''[|\omega| + 1] = \mathsf{pq}?\lambda$. This input matches the output $\tau''[1] = \mathsf{pq}!\lambda$. For inputs $\tau''[i]$ with $i > |\omega| + 1$, we know that $\tau''[i] = (\omega' \cdot \tau')[i-2]$, where $(\omega' \cdot \tau')[i-2]$ matches some output $(\omega' \cdot \tau')[j]$ in $\omega' \cdot \tau'$. Then $\tau''[i]$ matches $\tau''[j+1]$ if $j \leq |\omega'|$ and $\tau''[j+2]$ otherwise. This proves that $\omega \cdot \tau$ is well formed. □

We have now enough machinery to define events of simple agts, which are equivalence classes of pairs whose first element is an o-trace ω (representing the queue) and whose second element is a trace τ (representing a path in the sgt component of the agt). The traces ω and τ are considered respectively modulo $\cong$ and modulo $\approx_\omega$. The trace τ is ω-well formed, reflecting the input/output matching of sgts with respect to queues. The communication represented by an event is the last communication of τ.

Definition 7.12 (Global events)**.** (1) *The equivalence $\sim$ on pairs (ω, τ) where $\tau \neq \epsilon$ is ω-pointed is the least equivalence such that*

$$(\omega, \tau) \sim (\omega', \tau') \text{ if } \omega \cong \omega' \text{ and } \tau \approx_\omega \tau'$$

(2) *A* g-event *$\delta = [\omega, \tau]_\sim$ is the equivalence class of the pair (ω, τ). The communication of δ, notation $\mathsf{i/o}(\delta)$, is defined to be $\mathsf{last}(\tau)$.*
(3) *We denote by $\mathcal{GE}$ the set of g-events.*

Notice that the function $\mathsf{i/o}$ can be applied both to an n-event (Definition 5.3(2)) and to a g-event (Definition 7.12(2)). In all cases the result is a communication.

Given an o-trace ω and an arbitrary trace τ, we want to build a g-event $[\omega, \tau']_\sim$ (Definition 7.14). To this aim we scan τ from right to left and remove all and only the communications $\tau[i]$ which violate the pointedness property.

Definition 7.13 (Trace filtering)**.** *The* filtering of $\tau \cdot \tau'$ by ω with cursor at τ, *denoted by* $\tau \lceil_\omega \tau'$, *is defined by induction on* τ *as follows:*

$$\epsilon \lceil_\omega \tau' = \tau' \qquad\qquad (\tau'' \cdot \beta) \lceil_\omega \tau' = \begin{cases} \tau'' \lceil_\omega (\beta \cdot \tau') & \text{if } \beta \cdot \tau' \text{ is } (\omega \cdot \tau'')\text{-pointed} \\ \tau'' \lceil_\omega \tau' & \text{otherwise} \end{cases}$$

For example $\mathsf{pq}?\lambda \cdot \mathsf{qp}?\lambda \lceil_{\mathsf{pq}!\lambda} \epsilon = \mathsf{pq}?\lambda \lceil_{\mathsf{pq}!\lambda} \epsilon = \epsilon \lceil_{\mathsf{pq}!\lambda} \mathsf{pq}?\lambda = \mathsf{pq}?\lambda$. The resulting trace can also be empty, as in $\mathsf{qp}?\lambda \lceil_{\mathsf{pq}!\lambda} \epsilon = \epsilon \lceil_{\mathsf{pq}!\lambda} \epsilon = \epsilon$. It is easy to verify that $\tau \lceil_\omega \tau'$ is a subtrace of $\tau \cdot \tau'$, and that if τ is ω-pointed, then $\tau \lceil_\omega \epsilon = \tau$.

Definition 7.14 (G-event of a pair)**.** *Let* $\tau \neq \epsilon$ *be* ω*-well formed. The* g-event *generated by* ω *and* τ*, notation* $\mathsf{ev}(\omega, \tau)$*, is defined to be* $\mathsf{ev}(\omega, \tau) = [\omega, \tau \lceil_\omega \epsilon]_\sim$.

Hence the trace of the event $\mathsf{ev}(\omega, \tau)$ is the filtering of τ by ω with cursor at the end of τ. This definition is sound since $\omega \cong \omega'$ implies $\tau \lceil_\omega \tau' = \tau \lceil_{\omega'} \tau'$. Moreover $\mathsf{ev}(\omega, \tau)$ enjoys a useful property, as shown by the following lemma.

Lemma 7.15 . *If* $\mathsf{ev}(\omega, \tau)$ *is defined, then* $\tau \lceil_\omega \epsilon \neq \epsilon$ *and* $\mathsf{i/o}(\mathsf{ev}(\omega, \tau)) = \mathsf{last}(\tau \lceil_\omega \epsilon) = \mathsf{last}(\tau)$.

Proof. Let $\tau \neq \epsilon$ be ω-well formed and $\tau = \tau' \cdot \beta$. Then β is $(\omega \cdot \tau')$-well formed by Definition 7.2. This implies that β is $(\omega \cdot \tau')$-pointed by Definition 7.7, and thus $\tau \lceil_\omega \epsilon = (\tau' \cdot \beta) \lceil_\omega \epsilon = \tau' \lceil_\omega \beta$. This gives $\mathsf{i/o}(\mathsf{ev}(\omega, \tau)) = \mathsf{last}(\tau \lceil_\omega \epsilon) = \mathsf{last}(\tau)$. □

Since the o-traces in the g-events of a simple agt correspond to the message queue, we define the causality and conflict relations only between g-events with the same o-traces. Causality is then simply prefixing of traces, while conflict is induced by the conflict relation on the p-events obtained by projecting the traces on participants (Definition 5.4(1)).

Definition 7.16 (Causality and conflict relations on g-events)**.** *The* causality *relation* $\leq$ *and the* conflict *relation* $\#$ *on the set of g-events* $\mathcal{GE}$ *are defined by:*

(1) $[\omega, \tau]_\sim \leq [\omega, \tau']_\sim$ *if* $\tau' \approx_\omega \tau \cdot \tau_1$ *for some* τ_1*;*
(2) $[\omega, \tau]_\sim \# [\omega, \tau']_\sim$ *if* $\tau \mathbin{@} \mathsf{p} \# \tau' \mathbin{@} \mathsf{p}$ *for some* p.

Concerning Clause (1), note that the relation $\leq$ is able to express cross-causality as well as local causality, thanks to the hypothesis of ω-well formedness of τ in any g-event $[\omega, \tau]_\sim$. Indeed, this hypothesis implies that, whenever τ ends by an input $\mathsf{pq}?\lambda$, then the matched output $\mathsf{pq}!\lambda$ must appear either in ω, in which case the output has already occurred, or at some position i in τ. In the latter case, the g-event $\mathsf{ev}(\omega, \tau[1 \ldots i])$, which represents the output $\mathsf{pq}!\lambda$, is such that $\mathsf{ev}(\omega, \tau[1 \ldots i]) \leq [\omega, \tau]_\sim$. In fact, the statement $\mathsf{ev}(\omega, \tau[1 \ldots i]) \leq [\omega, \tau]_\sim$ holds in general, also when $\tau[i]$ and $\tau[n]$ are not a pair of communications in the matching relation, as proved in Lemma 7.17.

As regards Clause (2), note that if $\tau \approx_\omega \tau'$, then $\tau \mathbin{@} \mathsf{p} = \tau' \mathbin{@} \mathsf{p}$ for all p, because $\approx_\omega$ does not swap communications with the same player. Hence, conflict is well defined, since it does not depend on the trace chosen in the equivalence class. The condition $\tau \mathbin{@} \mathsf{p} \# \tau' \mathbin{@} \mathsf{p}$ states that participant p does the same actions in both traces up to some point, after which it performs two different actions in τ and τ'.

Lemma 7.17 . *Let* $[\omega, \tau]_\sim$ *be a g-event. Then* $\mathsf{ev}(\omega, \tau[1 \dots i]) \leq [\omega, \tau]_\sim$.

Proof. If $[\omega, \tau]_\sim$ is a g-event, then $\tau[1 \dots i]$ is ω-well formed by Definition 7.12, which implies that $\mathsf{ev}(\omega, \tau[1 \dots i])$ is defined. Let $\tau[i] = \beta$, then $\tau[1 \dots i] \lceil_\omega \epsilon = \tau' \cdot \beta$ for some τ' by Lemma 7.15. We want to show that $\tau \approx_\omega \tau' \cdot \beta \cdot \tau_0$ for some τ_0. By Definition 7.13 τ' is a subtrace of $\tau[1 \dots i-1]$. Assume $\tau' = \tau_1 \cdot \ldots \cdot \tau_k$ and $\tau[1 \dots i-1] = \tau_1 \cdot \beta_2 \cdot \ldots \cdot \beta_k \cdot \tau_k$, where $k \geq 1$. If $k = 1$ we are done. Otherwise, the filtering of $\tau_1 \cdot \beta_2 \cdot \ldots \cdot \beta_k \cdot \tau_k \cdot \beta$ by ω with cursor at $\tau_1 \cdot \beta_2 \cdot \ldots \cdot \beta_k$ erases β_k. I. e. $(\tau_1 \cdot \beta_2 \cdot \ldots \cdot \tau_{k-1} \cdot \beta_k) \lceil_\omega \tau_k \cdot \beta = (\tau_1 \cdot \beta_2 \cdot \ldots \cdot \tau_{k-1}) \lceil_\omega \tau_k \cdot \beta$. This means that $\beta_k \cdot \tau_k \cdot \beta$ is not $(\omega \cdot \tau_1 \cdot \beta_2 \cdot \ldots \cdot \tau_{k-1})$-pointed by Definition 7.13. Instead $\tau_k \cdot \beta$ is $(\omega \cdot \tau_1 \cdot \beta_2 \cdot \ldots \cdot \tau_{k-1} \cdot \beta_k)$-pointed. So the failure of the pointedness property is due to β_k. By Definitions 7.7 and 7.3 $\tau_1 \cdot \beta_2 \cdot \ldots \cdot \beta_k \cdot \tau_k \cdot \beta$ ω-swaps (in several steps) to $\tau_1 \cdot \beta_2 \cdot \ldots \cdot \tau_k \cdot \beta \cdot \beta_k$. By iterating this argument for all $\beta_j, 2 \leq j < k$, we obtain $\tau[1 \dots i] \approx_\omega \tau_1 \cdot \ldots \cdot \tau_k \cdot \beta \cdot \beta_2 \cdot \ldots \cdot \beta_k$. We conclude $\tau = \tau[1 \dots i] \cdot \tau[i+1 \dots n] \approx_\omega \tau' \cdot \beta \cdot \beta_2 \cdot \ldots \cdot \beta_k \cdot \tau[i+1 \dots n]$. □

We get the events of a simple agt $\mathsf{G} \parallel \mathcal{M}$ by applying the function ev to the pairs made of the o-trace of the queue $\mathcal{M}$ and a trace in the tree of G. Lemma 7.11 and Definition 7.14 ensure that ev is defined. We then build the ES associated with a simple agt $\mathsf{G} \parallel \mathcal{M}$ as follows.

Definition 7.18 (Event Structure of a Global Type)**.** *The* event structure of the simple agt $\mathsf{G} \parallel \mathcal{M}$ *is the triple*

$$\mathcal{S}^{\mathcal{G}}(\mathsf{G} \parallel \mathcal{M}) = (\mathcal{GE}(\mathsf{G} \parallel \mathcal{M}), \leq_{\mathsf{G}\parallel\mathcal{M}}, \#_{\mathsf{G}\parallel\mathcal{M}})$$

where:

(1) $\mathcal{GE}(\mathsf{G} \parallel \mathcal{M}) = \{\mathsf{ev}(\omega, \tau) \mid \omega = \mathsf{otr}(\mathcal{M}) \,\&\, \tau \in \mathsf{Tr}^+(\mathsf{G})\}$;
(2) $\leq_{\mathsf{G}\parallel\mathcal{M}}$ *is the restriction of* $\leq$ *to the set* $\mathcal{GE}_{\mathcal{S}}(\mathsf{G} \parallel \mathcal{M})$;
(3) $\#_{\mathsf{G}\parallel\mathcal{M}}$ *is the restriction of* $\#$ *to the set* $\mathcal{GE}_{\mathcal{S}}(\mathsf{G} \parallel \mathcal{M})$.

Example 7.19 . (1) *Let* $\mathsf{G} = \mathsf{pq}!\lambda; \mathsf{qp}!\lambda'; \mathsf{qp}?\lambda'; \mathsf{pq}?\lambda$. *Then* $\mathcal{GE}(\mathsf{G} \parallel \emptyset) = \{\delta_1, \dots, \delta_4\}$ *where* $\delta_1 = [\epsilon, \mathsf{pq}!\lambda]_\sim$, $\delta_2 = [\epsilon, \mathsf{qp}!\lambda']_\sim$, $\delta_3 = [\epsilon, \mathsf{pq}!\lambda \cdot \mathsf{qp}!\lambda' \cdot \mathsf{qp}?\lambda']_\sim$, $\delta_4 = [\epsilon, \mathsf{pq}!\lambda \cdot \mathsf{qp}!\lambda' \cdot \mathsf{pq}?\lambda]_\sim$.
(2) *Let* $\mathsf{G}_1 = \mathsf{pq}!\lambda; \mathsf{pr}!\lambda'; \mathsf{pq}?\lambda; \mathsf{pr}?\lambda'$. *Then* $\mathcal{GE}(\mathsf{G}_1 \parallel \emptyset) = \{\delta_1, \dots, \delta_4\}$ *where* $\delta_1 = [\epsilon, \mathsf{pq}!\lambda]_\sim$, $\delta_2 = [\epsilon, \mathsf{pq}!\lambda \cdot \mathsf{pr}!\lambda']_\sim$, $\delta_3 = [\epsilon, \mathsf{pq}!\lambda \cdot \mathsf{pq}?\lambda]_\sim$, $\delta_4 = [\epsilon, \mathsf{pq}!\lambda \cdot \mathsf{pr}!\lambda' \cdot \mathsf{pr}?\lambda']_\sim$.
(3) *Let* $\mathsf{G}_2 = \mathsf{pq}!\lambda; \mathsf{pr}!\lambda'; \mathsf{pr}?\lambda'; \mathsf{pq}?\lambda$. *Then* $\mathcal{GE}(\mathsf{G}_2 \parallel \emptyset) = \mathcal{GE}(\mathsf{G}_1 \parallel \emptyset)$.
(4) *Let* $\mathsf{G}'_1 = \mathsf{pq}?\lambda; \mathsf{pr}?\lambda'$. *Then* $\mathcal{GE}(\mathsf{G}'_1 \parallel \langle \mathsf{p}, \lambda, \mathsf{q} \rangle \cdot \langle \mathsf{p}, \lambda', \mathsf{r} \rangle) = \{\delta_1, \delta_2\}$ *where* $\delta_1 = [\mathsf{pq}!\lambda \cdot \mathsf{pr}!\lambda', \mathsf{pq}?\lambda]_\sim$ *and* $\delta_2 = [\mathsf{pq}!\lambda \cdot \mathsf{pr}!\lambda', \mathsf{pr}?\lambda']_\sim$.
(5) *Let* $\mathsf{G}'_2 = \mathsf{pr}?\lambda'; \mathsf{pq}?\lambda$. *Then* $\mathcal{GE}(\mathsf{G}'_2 \parallel \langle \mathsf{p}, \lambda, \mathsf{q} \rangle \cdot \langle \mathsf{p}, \lambda', \mathsf{r} \rangle) = \mathcal{GE}(\mathsf{G}'_1 \parallel \langle \mathsf{p}, \lambda, \mathsf{q} \rangle \cdot \langle \mathsf{p}, \lambda', \mathsf{r} \rangle)$.

The following example shows that, due to the possibility of anticipating communications that are independent from a choice and to the fact that sgts are not able to represent concurrency explicitly, two diverging traces in the tree representation of G do not necessarily give rise to two conflicting events in $\mathcal{GE}(\mathsf{G} \parallel \mathcal{M})$.

Example 7.20 . *Let* $\mathsf{G} = \mathsf{pq}!\lambda; \mathsf{rs}!(\lambda_1; \mathsf{pq}?\lambda; \mathsf{rs}?\lambda_1 + \lambda_2; \mathsf{pq}?\lambda; \mathsf{rs}?\lambda_2)$. *Then* $\mathcal{GE}(\mathsf{G} \parallel \emptyset)$ *contains the g-event* $[\epsilon, \mathsf{pq}!\lambda \cdot \mathsf{pq}?\lambda]_\sim$ *generated by the two diverging traces of* G*:*

$$\mathsf{pq}!\lambda \cdot \mathsf{rs}!\lambda_1 \cdot \mathsf{pq}?\lambda \qquad\qquad \mathsf{pq}!\lambda \cdot \mathsf{rs}!\lambda_2 \cdot \mathsf{pq}?\lambda$$

Note on the other hand that if we replace r *by* q *in* G*, namely if we consider the sgt* $\mathsf{G}' = \mathsf{pq}!\lambda; \mathsf{qs}!(\lambda_1; \mathsf{pq}?\lambda; \mathsf{qs}?\lambda_1 + \lambda_2; \mathsf{pq}?\lambda; \mathsf{qs}?\lambda_2)$*, then* $\mathcal{GE}(\mathsf{G}' \parallel \emptyset)$ *contains* $\delta = [\epsilon, \mathsf{pq}!\lambda \cdot \mathsf{qs}!\lambda_1 \cdot \mathsf{pq}?\lambda]_\sim$ *and* $\delta' = [\epsilon, \mathsf{pq}!\lambda \cdot \mathsf{qs}!\lambda_2 \cdot \mathsf{pq}?\lambda]_\sim$*. Here* $\delta \# \delta'$ *because*

$$(\mathsf{pq}!\lambda \cdot \mathsf{qs}!\lambda_1 \cdot \mathsf{pq}?\lambda) @ \mathsf{q} = \mathsf{s}!\lambda_1 \cdot \mathsf{p}?\lambda \# \mathsf{s}!\lambda_2 \cdot \mathsf{p}?\lambda = (\mathsf{pq}!\lambda \cdot \mathsf{qs}!\lambda_2 \cdot \mathsf{pq}?\lambda) @ \mathsf{q}$$

So, here the two occurrences of $\mathsf{pq}?\lambda$ *are represented by two distinct events that are are in conflict.*

We end this section showing that the obtained ES is a PES.

Proposition 7.21 . *Let* $\mathsf{G} \parallel \mathcal{M}$ *be a simple agt. Then* $\mathcal{S}^{\mathcal{G}}(\mathsf{G} \parallel \mathcal{M})$ *is a prime event structure.*

Proof. We show that $\leq$ and $\#$ satisfy Properties (2) and (3) of Definition 3.1. Reflexivity and transitivity of $\leq$ follow easily from the properties of concatenation and the properties of the two equivalences in Definitions 5.2 and 7.5. As for antisymmetry note that, by Clause (1) of Definition 7.16, if $[\omega, \tau]_\sim \leq [\omega, \tau']_\sim$ and $[\omega, \tau']_\sim \leq [\omega, \tau]_\sim$, then $\tau \cdot \tau_1 \approx_\omega \tau'$ and $\tau' \cdot \tau_2 \approx_\omega \tau$ for some τ_1 and τ_2. Hence $\tau \cdot \tau_1 \cdot \tau_2 \approx_\omega \tau$, which implies $\tau_1 = \tau_2 = \epsilon$, i.e. $\tau \approx_\omega \tau'$.
The conflict between g-events inherits irreflexivity, symmetry and hereditariness from the conflict between p-events. In particular, for hereditariness, suppose that $[\omega, \tau]_\sim \# [\omega, \tau']_\sim \leq [\omega, \tau'']_\sim$. Then $\tau'' \approx_\omega \tau' \cdot \tau_1$ for some τ_1 and $\tau'' \,@\, \mathsf{p} = (\tau' \cdot \tau_1) \,@\, \mathsf{p} = (\tau' \,@\, \mathsf{p}) \cdot (\tau_1 \,@\, \mathsf{p}) \# \tau \,@\, \mathsf{p}$ since $\tau' \,@\, \mathsf{p} \# \tau \,@\, \mathsf{p}$. □

8. Equivalence of the two Event Structure Semantics

In this section we establish our main theorem for typed networks, namely the isomorphism between the configuration domain of the FES of the network and the configuration domain of the PES of its simple agt. To prove the various results leading to this theorem, we will largely use the characterisation of configurations as proving sequences, as given in Proposition 3.7. Let us briefly sketch how these results are articulated.

The proof of the isomorphism is grounded on the Subject Reduction Theorem (Theorem 6.18) and the Session Fidelity Theorem (Theorem 6.19). These theorems state that if $\vdash \mathsf{N} \parallel \mathcal{M} : \mathsf{G} \parallel \mathcal{M}$, then $\mathsf{N} \parallel \mathcal{M} \xrightarrow{\tau} \mathsf{N}' \parallel \mathcal{M}'$ if and only if $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\tau} \mathsf{G}' \parallel \mathcal{M}'$, and in both directions $\vdash \mathsf{N}' \parallel \mathcal{M}' : \mathsf{G}' \parallel \mathcal{M}'$. We can then relate the ESs of networks and simple agts by connecting them through the traces of their transition sequences, and by taking into account the queues by means of the mapping otr given by Definition 5.1. This is achieved as follows.

If $\mathsf{N} \parallel \mathcal{M} \xrightarrow{\tau}$ and $\mathsf{otr}(\mathcal{M}) = \omega$, then the function nec (Definition 8.8) applied to ω and τ gives a proving sequence in $\mathcal{S}^{\mathcal{N}}(\mathsf{N} \parallel \mathcal{M})$ (Theorem 8.11). Vice versa, if $\rho_1; \cdots; \rho_n$ is a proving sequence in $\mathcal{S}^{\mathcal{N}}(\mathsf{N} \parallel \mathcal{M})$, then $\mathsf{N} \parallel \mathcal{M} \xrightarrow{\tau} \mathsf{N}' \parallel \mathcal{M}'$ where $\tau = \mathsf{i/o}(\rho_1) \cdots \mathsf{i/o}(\rho_n)$ and $\mathsf{i/o}$ is the mapping given in Definition 5.3(2) (Theorem 8.12).

Similarly, if $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\tau} \mathsf{G}' \parallel \mathcal{M}'$ and $\mathsf{otr}(\mathcal{M}) = \omega$, then the function gec (Definition 8.21) applied to ω and τ gives a proving sequence in $\mathcal{S}^{\mathcal{G}}(\mathsf{G} \parallel \mathcal{M})$ (Theorem 8.26). Lastly, if $\delta_1; \ldots; \delta_n$ is a proving sequence in $\mathcal{S}^{\mathcal{G}}(\mathsf{G} \parallel \mathcal{M})$, then $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\tau} \mathsf{G}' \parallel \mathcal{M}'$, where $\tau = \mathsf{i/o}(\delta_1) \cdot \ldots \cdot \mathsf{i/o}(\delta_n)$ and $\mathsf{i/o}$ is the mapping given in Definition 7.12(2) (Theorem 8.27).

It is then natural to split this section in three subsections: the first establishing the relationship between network transition sequences and proving sequences of their event structure, the second doing the same for simple agts and finally a third subsection in which the isomorphism between the two configuration domains is proved relying on these relationships.

8.1. **Transition Sequences of Networks and Proving Sequences of their ESs.**

We start by showing how network communications affect n-events in the associated ES. To this aim we define two partial operators $\lozenge$ and $\blacklozenge$, which applied to a communication β

and an n-event ρ yield another n-event ρ' (when defined). The intuition is that ρ' represents the event ρ as it was before the communication β, or as it will be after the communication β, respectively. So, in particular, if ρ is not located at $\mathsf{play}(\beta)$, its p-event will remain unchanged under both mappings $\lozenge$ and $\blacklozenge$, and only its queue will be affected. We shall now explain in more detail how these operators work.

The operator $\lozenge$, when applied to β and ρ, yields the n-event ρ' obtained from ρ before executing the communication β, if it exists. We call $\beta \lozenge \rho$ the *retrieval* of ρ before β. So, if $\beta = \mathsf{pq}!\lambda$ and the queue of ρ ends with $\mathsf{pq}!\lambda$, the queue of ρ' is obtained by removing $\mathsf{pq}!\lambda$ from the end of the queue of ρ. Moreover, if ρ is located at p, the p-event of ρ' is obtained by adding $\mathsf{q}!\lambda$ in front of the p-event of ρ. If $\beta = \mathsf{pq}?\lambda$, the queue of ρ' is obtained by adding $\mathsf{pq}!\lambda$ on top of the queue of ρ. Moreover, if ρ is located at q, the p-event of ρ' is obtained by adding $\mathsf{q}?\lambda$ in front of the p-event of ρ.

The operator $\blacklozenge$, when applied to β and ρ, yields the n-event ρ' obtained from ρ after executing the communication β, if this event exists. We call $\beta \blacklozenge \rho$ the *residual* of ρ after β. So, if $\beta = \mathsf{pq}!\lambda$ the queue of ρ' is obtained from the queue of ρ by enqueuing $\mathsf{pq}!\lambda$. Moreover, if ρ is located at p and its p-event starts with the action $\mathsf{q}!\lambda$, then the p-event of ρ' is obtained by removing this action, provided the result is still a p-event (this will not happen if the p-event of ρ is a simple action, because in this case it will be consumed by the communication β); otherwise, the operation is not defined for ρ located at p. If $\beta = \mathsf{pq}?\lambda$ and the queue of ρ starts with $\mathsf{pq}!\lambda$, the queue of ρ' is obtained by dequeuing $\mathsf{pq}!\lambda$. Moreover, if ρ is located at q and its p-event starts with the action $\mathsf{p}?\lambda$, the p-event of ρ' is obtained by removing $\mathsf{p}?\lambda$, if possible; otherwise, the operation is not defined.

Definition 8.1 (Retrieval and residual of an n-event with respect to a communication)**.**

(1) *(Retrieval of an n-event before a communication) The operator $\lozenge$ applied to a communication β and an n-event ρ is defined by cases on β:*

$$\mathsf{pq}!\lambda \lozenge (\mathsf{r} :: ((\omega \cdot \mathsf{pq}!\lambda)_{\cong}, \eta)) = \begin{cases} \mathsf{r} :: (\omega_{\cong}, \mathsf{q}!\lambda \cdot \eta) & \textit{if } \mathsf{r} = \mathsf{p} \\ \mathsf{r} :: (\omega_{\cong}, \eta) & \textit{otherwise} \end{cases}$$

$$\mathsf{pq}?\lambda \lozenge (\mathsf{r} :: (\omega_{\cong}, \eta)) = \begin{cases} \mathsf{r} :: ((\mathsf{pq}!\lambda \cdot \omega)_{\cong}, \mathsf{p}?\lambda \cdot \eta) & \textit{if } \mathsf{r} = \mathsf{q} \\ \mathsf{r} :: ((\mathsf{pq}!\lambda \cdot \omega)_{\cong}, \eta) & \textit{otherwise} \end{cases}$$

(2) *(Residual of an n-event after a communication) The operator $\blacklozenge$ applied to a communication β and an n-event ρ is defined by cases on β:*

$$\mathsf{pq}!\lambda \blacklozenge (\mathsf{r} :: (\omega_{\cong}, \eta)) = \begin{cases} \mathsf{r} :: ((\omega \cdot \mathsf{pq}!\lambda)_{\cong}, \eta') & \textit{if } \mathsf{r} = \mathsf{p} \textit{ and } \eta = \mathsf{q}!\lambda \cdot \eta' \\ \mathsf{r} :: ((\omega \cdot \mathsf{pq}!\lambda)_{\cong}, \eta) & \textit{if } \mathsf{r} \neq \mathsf{p} \end{cases}$$

$$\mathsf{pq}?\lambda \blacklozenge (\mathsf{r} :: ((\mathsf{pq}!\lambda \cdot \omega)_{\cong}, \eta)) = \begin{cases} \mathsf{r} :: (\omega_{\cong}, \eta') & \textit{if } \mathsf{r} = \mathsf{q} \textit{ and } \eta = \mathsf{p}?\lambda \cdot \eta' \\ \mathsf{r} :: (\omega_{\cong}, \eta) & \textit{if } \mathsf{r} \neq \mathsf{q} \end{cases}$$

Notice that in Clause (2) of the above definition $\eta' \neq \epsilon$, see Definition 4.1. Observe also that the operators $\lozenge$ and $\blacklozenge$ preserve the communication of n-events, namely $\mathsf{i/o}(\beta \lozenge \rho) = \mathsf{i/o}(\beta \blacklozenge \rho) = \mathsf{i/o}(\rho)$.

The retrieval and residual operators on n-events induce (partial) mappings on o-traces, which it is handy to define explicitly.

Definition 8.2 . *The partial mappings $\beta \triangleright \omega$ and $\beta \blacktriangleright \omega$ are defined by:*

(1) $\mathsf{pq}!\lambda \triangleright \omega \cdot \mathsf{pq}!\lambda = \omega$ *and* $\mathsf{pq}?\lambda \triangleright \omega = \mathsf{pq}!\lambda \cdot \omega$*;*

(2) $\mathsf{pq}!\lambda \blacktriangleright \omega = \omega \cdot \mathsf{pq}!\lambda$ *and* $\mathsf{pq}?\lambda \blacktriangleright \mathsf{pq}!\lambda \cdot \omega = \omega$*.*

These mappings enjoy the following commutativity properties. In their proofs and elsewhere we shall use the *complement* $\overline{\beta}$ of an input β defined by $\overline{\beta} = \mathsf{pq}!\lambda$ if $\beta = \mathsf{pq}?\lambda$.

Lemma 8.3 . *Let* $\mathsf{play}(\beta_1) \cap \mathsf{play}(\beta_2) = \emptyset$.

(1) *If both* $\beta_1 \triangleright \omega$ *and* $\beta_2 \triangleright \omega$ *are defined, then* $\beta_1 \triangleright (\beta_2 \triangleright \omega)$ *is defined and* $\beta_1 \triangleright (\beta_2 \triangleright \omega) \cong \beta_2 \triangleright (\beta_1 \triangleright \omega)$.
(2) *If both* $\beta_2 \blacktriangleright \omega$ *and* $\beta_2 \blacktriangleright (\beta_1 \triangleright \omega)$ *are defined, then* $\beta_1 \triangleright (\beta_2 \blacktriangleright \omega) \cong \beta_2 \blacktriangleright (\beta_1 \triangleright \omega)$.

Proof. (1) Since $\omega_i = \beta_i \triangleright \omega$ is defined for $i \in \{1, 2\}$, by Definition 8.2(1) $\omega \cong \omega_i \cdot \beta_i$ when β_i is an output. Then from $\mathsf{play}(\beta_1) \cap \mathsf{play}(\beta_2) = \emptyset$ we get $\omega \cong \omega' \cdot \beta_1 \cdot \beta_2 \cong \omega' \cdot \beta_2 \cdot \beta_1$ for some ω' when both β_1 and β_2 are outputs. Using Definition 8.2(1) we compute:

$$\beta_1 \triangleright (\beta_2 \triangleright \omega) \cong \beta_2 \triangleright (\beta_1 \triangleright \omega) \cong \begin{cases} \omega' & \text{if both } \beta_1 \text{ and } \beta_2 \text{ are outputs} \\ \overline{\beta_i} \cdot \omega_j & \text{if } \beta_i \text{ is an input and } \beta_j \text{ is an output} \\ \overline{\beta_1} \cdot \overline{\beta_2} \cdot \omega & \text{if both } \beta_1 \text{ and } \beta_2 \text{ are inputs} \end{cases}$$

(2) Since $\omega_2 = \beta_2 \blacktriangleright \omega$ is defined, by Definition 8.2(2) $\omega \cong \overline{\beta_2} \cdot \omega_2$ when β_2 is an input. Since $\beta_2 \blacktriangleright (\beta_1 \triangleright \omega)$ is defined, $\omega_1 = \beta_1 \triangleright \omega$ is defined and by Definition 8.2 $\omega \cong \omega_1 \cdot \beta_1$ when β_1 is an output and $\omega \cong \overline{\beta_2} \cdot \omega_0 \cdot \beta_1$ for some ω_0 such that $\omega_1 \cong \overline{\beta_2} \cdot \omega_0$ and $\omega_2 \cong \omega_0 \cdot \beta_1$, when β_1 is an output and β_2 is an input. Using Definition 8.2 we compute:

$$\beta_1 \triangleright (\beta_2 \blacktriangleright \omega) \cong \beta_2 \blacktriangleright (\beta_1 \triangleright \omega) \cong \begin{cases} \omega_1 \cdot \beta_2 & \text{if both } \beta_1 \text{ and } \beta_2 \text{ are outputs} \\ \omega_0 & \text{if } \beta_1 \text{ is an output and } \beta_2 \text{ is an input} \\ \overline{\beta_1} \cdot \omega \cdot \beta_2 & \text{if } \beta_1 \text{ is an input and } \beta_2 \text{ is an output} \\ \overline{\beta_1} \cdot \omega_2 & \text{if both } \beta_1 \text{ and } \beta_2 \text{ are inputs} \end{cases}$$

□

Notice that the facts that both $\beta_1 \triangleright \omega$ and $\beta_2 \blacktriangleright \omega$ are defined and $\mathsf{play}(\beta_1) \cap \mathsf{play}(\beta_2) = \emptyset$ do not imply that $\beta_2 \blacktriangleright (\beta_1 \triangleright \omega)$ is defined. As an example, consider the case where β_1 is an output, β_2 is the complementary input, i.e. $\overline{\beta_2} = \beta_1$, and $\omega = \beta_1$, which gives $\beta_1 \triangleright \omega = \beta_2 \blacktriangleright \omega = \epsilon$. The problem here is that $\beta_1 \triangleright \omega$ and $\beta_2 \blacktriangleright \omega$ consume the same output β_1 in the queue ω, hence they are not independent.

Using the mappings $\beta \triangleright \omega$ and $\beta \blacktriangleright \omega$, we can write the retrieval and residual of n-events in a more concise (but less explicit) form, as stated in the following lemma which uses the projection of o-traces on participants (Definition 5.4(1)). This form simplifies some statements and proofs.

Lemma 8.4 . (1) *If* $\beta \lozenge \mathsf{r} :: (\omega_{\cong}, \eta)$ *is defined, then* $\beta \lozenge \mathsf{r} :: (\omega_{\cong}, \eta) = \mathsf{r} :: ((\beta \triangleright \omega)_{\cong}, \beta \mathbin{@} \mathsf{r} \cdot \eta)$.
(2) *If* $\beta \blacklozenge \mathsf{r} :: (\omega_{\cong}, \beta \mathbin{@} \mathsf{r} \cdot \eta)$ *is defined, then* $\beta \blacklozenge \mathsf{r} :: (\omega_{\cong}, \beta \mathbin{@} \mathsf{r} \cdot \eta) = \mathsf{r} :: ((\beta \blacktriangleright \omega)_{\cong}, \eta)$.

An immediate consequence of this lemma is that the retrieval and residual operators are inverse of each other.

Lemma 8.5 . (1) *If* $\beta \lozenge \rho$ *is defined, then* $\beta \blacklozenge (\beta \lozenge \rho) = \rho$.
(2) *If* $\beta \blacklozenge \rho$ *is defined, then* $\beta \lozenge (\beta \blacklozenge \rho) = \rho$.

Moreover, the retrieval and residual operators preserve the flow and conflict relations.

Lemma 8.6 . (1) *If* $\rho \prec \rho'$ *and* $\beta \lozenge \rho$ *is defined, then* $\beta \lozenge \rho \prec \beta \lozenge \rho'$.
(2) *If* $\rho \prec \rho'$ *and both* $\beta \blacklozenge \rho$ *and* $\beta \blacklozenge \rho'$ *are defined, then* $\beta \blacklozenge \rho \prec \beta \blacklozenge \rho'$.
(3) *If* $\rho \# \rho'$ *and* $\beta \lozenge \rho$ *is defined, then* $\beta \lozenge \rho \# \beta \lozenge \rho'$.
(4) *If* $\rho \# \rho'$ *and both* $\beta \blacklozenge \rho$ *and* $\beta \blacklozenge \rho'$ *are defined, then* $\beta \blacklozenge \rho \# \beta \blacklozenge \rho'$.

Proof. (1) Since $\rho \prec \rho'$ and $\beta \lozenge \rho$ is defined, also $\beta \lozenge \rho'$ is defined. If $\rho \prec \rho'$, then

– either $\rho = \mathsf{p} :: (\omega_{\cong}, \eta)$ and $\rho' = \mathsf{p} :: (\omega_{\cong}, \eta')$ and $\eta < \eta'$,
– or $\rho = \mathsf{p} :: (\omega_{\cong}, \zeta \cdot \mathsf{q}!\lambda)$ and $\rho' = \mathsf{q} :: (\omega_{\cong}, \zeta' \cdot \mathsf{p}?\lambda)$ and $(\omega \mathbin{@} \mathsf{p} \cdot \zeta)\mathbin{\Rsh}\mathsf{q} \lesssim \bowtie_{\gtrsim} (\omega \mathbin{@} \mathsf{q} \cdot \zeta'')\mathbin{\Rsh}\mathsf{p}$ for some ζ'' and χ such that $(\zeta' \cdot \mathsf{p}?\lambda)\mathbin{\Rsh}\mathsf{p} \precsim (\zeta'' \cdot \mathsf{p}?\lambda \cdot \chi)\mathbin{\Rsh}\mathsf{p}$.

In the first case by Lemma 8.4(1) it is easy to get the result.
In the second case we consider $\beta = \mathsf{pq}?\lambda'$, the proofs for other choices of β being similar and sometimes simpler. By Definition 8.1(1) $\beta \lozenge \rho = \mathsf{p} :: ((\mathsf{pq}!\lambda' \cdot \omega)_{\cong}, \zeta \cdot \mathsf{q}!\lambda)$ and $\beta \lozenge \rho' = \mathsf{q} :: ((\mathsf{pq}!\lambda' \cdot \omega)_{\cong}, \mathsf{p}?\lambda' \cdot \zeta' \cdot \mathsf{p}?\lambda)$. Notice that $(\mathsf{pq}!\lambda' \cdot \omega)\mathbin{@}\mathsf{p} = \mathsf{q}!\lambda' \cdot \omega \mathbin{@} \mathsf{p}$ and $(\mathsf{pq}!\lambda' \cdot \omega)\mathbin{@}\mathsf{q} = \omega \mathbin{@} \mathsf{q}$. Then, from $(\omega \mathbin{@} \mathsf{p} \cdot \zeta)\mathbin{\Rsh}\mathsf{q} \lesssim \bowtie_{\gtrsim} (\omega \mathbin{@} \mathsf{q} \cdot \zeta'')\mathbin{\Rsh}\mathsf{p}$ we get

$$(\mathsf{q}!\lambda' \cdot \omega \mathbin{@} \mathsf{p} \cdot \zeta)\mathbin{\Rsh}\mathsf{q} \lesssim \bowtie_{\gtrsim} (\omega \mathbin{@} \mathsf{q} \cdot \mathsf{p}?\lambda' \cdot \zeta'')\mathbin{\Rsh}\mathsf{p}$$

since $(\omega \mathbin{@} \mathsf{q} \cdot \mathsf{p}?\lambda' \cdot \zeta'')\mathbin{\Rsh}\mathsf{p} \precsim (\mathsf{p}?\lambda' \cdot \omega \mathbin{@} \mathsf{q} \cdot \zeta'')\mathbin{\Rsh}\mathsf{p}$. From $(\zeta' \cdot \mathsf{p}?\lambda)\mathbin{\Rsh}\mathsf{p} \precsim (\zeta'' \cdot \mathsf{p}?\lambda \cdot \chi)\mathbin{\Rsh}\mathsf{p}$ we get $(\mathsf{p}?\lambda' \cdot \zeta' \cdot \mathsf{p}?\lambda)\mathbin{\Rsh}\mathsf{p} \precsim (\mathsf{p}?\lambda' \cdot \zeta'' \cdot \mathsf{p}?\lambda \cdot \chi)\mathbin{\Rsh}\mathsf{p}$. We conclude $\beta \lozenge \rho \prec \beta \lozenge \rho'$.

(2) The proof is similar to that of Fact (1).

(3) Since $\rho \# \rho'$ and $\beta \lozenge \rho$ is defined, also $\beta \lozenge \rho'$ is defined. Let $\rho = \mathsf{p} :: (\omega_{\cong}, \eta)$ and $\rho' = \mathsf{p} :: (\omega_{\cong}, \eta')$ and $\eta \# \eta'$. Let $\beta = \mathsf{qp}?\lambda$, the proofs for other choices of β being similar and sometimes simpler. Letting $\omega' \cong \mathsf{qp}!\lambda \cdot \omega$, we get $\beta \lozenge \rho = \mathsf{p} :: (\omega'_{\cong}, \mathsf{q}?\lambda \cdot \eta)$ and $\beta \lozenge \rho' = \mathsf{p} :: (\omega'_{\cong}, \mathsf{q}?\lambda \cdot \eta')$. Since $\eta \# \eta'$ implies $\mathsf{q}?\lambda \cdot \eta \# \mathsf{q}?\lambda \cdot \eta'$, we conclude $\beta \lozenge \rho \# \beta \lozenge \rho'$.

(4) The proof is similar to that of Fact (3). □

We now show that the operators $\lozenge$ and $\blacklozenge$ applied to a communication β modify the n-events of an ES in the same way as the (backward or forward) execution of β would do in the underlying network.

Lemma 8.7 . *Let* $\mathsf{N} \parallel \mathcal{M} \xrightarrow{\beta} \mathsf{N}' \parallel \mathcal{M}'$. *Then* $\mathsf{otr}(\mathcal{M}) \cong \beta \triangleright \mathsf{otr}(\mathcal{M}')$ *and*
(1) *if* $\rho \in \mathcal{NE}(\mathsf{N}' \parallel \mathcal{M}')$, *then* $\beta \lozenge \rho \in \mathcal{NE}(\mathsf{N} \parallel \mathcal{M})$;
(2) *if* $\rho \in \mathcal{NE}(\mathsf{N} \parallel \mathcal{M})$ *and* $\beta \blacklozenge \rho$ *is defined, then* $\beta \blacklozenge \rho \in \mathcal{NE}(\mathsf{N}' \parallel \mathcal{M}')$.

Proof. Let $\beta = \mathsf{pq}!\lambda$. From $\mathsf{N} \parallel \mathcal{M} \xrightarrow{\beta} \mathsf{N}' \parallel \mathcal{M}'$ we get $\mathsf{p}[\![\bigoplus_{i\in I} \mathsf{q}!\lambda_i; P_i]\!] \in \mathsf{N}$ and $\mathsf{p}[\![P_k]\!] \in \mathsf{N}'$ with $\lambda = \lambda_k$ for some $k \in I$ and $\mathcal{M}' \equiv \mathcal{M} \cdot \langle \mathsf{p}, \lambda, \mathsf{q} \rangle$ and $\mathsf{r}[\![R_\mathsf{r}]\!] \in \mathsf{N}$ iff $\mathsf{r}[\![R_\mathsf{r}]\!] \in \mathsf{N}'$ for all $\mathsf{r} \neq \mathsf{p}$.
Let $\beta = \mathsf{pq}?\lambda$. From $\mathsf{N} \parallel \mathcal{M} \xrightarrow{\beta} \mathsf{N}' \parallel \mathcal{M}'$ we get $\mathsf{q}[\![\sum_{i\in I} \mathsf{p}?\lambda_i; Q_i]\!] \in \mathsf{N}$ and $\mathsf{q}[\![Q_k]\!] \in \mathsf{N}'$ with $\lambda = \lambda_k$ for some $k \in I$ and $\mathcal{M} \equiv \langle \mathsf{p}, \lambda, \mathsf{q} \rangle \cdot \mathcal{M}'$ and $\mathsf{r}[\![R_\mathsf{r}]\!] \in \mathsf{N}$ iff $\mathsf{r}[\![R_\mathsf{r}]\!] \in \mathsf{N}'$ for all $\mathsf{r} \neq \mathsf{q}$.
In both cases we get $\mathsf{otr}(\mathcal{M}) \cong \beta \triangleright \mathsf{otr}(\mathcal{M}')$.

(1) Assume $\rho = \mathsf{r} :: (\omega_{\cong}, \eta) \in \mathcal{NE}(\mathsf{N}' \parallel \mathcal{M}')$. By Definition 5.8(1) we have $\omega = \mathsf{otr}(\mathcal{M}')$. Let $\omega' = \mathsf{otr}(\mathcal{M}) \cong \beta \triangleright \omega$.
Let $\beta = \mathsf{pq}!\lambda$. By Definition 5.8(1) we have $\eta \in \mathcal{PE}(P_k)$ if $\mathsf{r} = \mathsf{p}$ and $\eta \in \mathcal{PE}(R_\mathsf{r})$ if $\mathsf{r} \neq \mathsf{p}$. By Definition 8.1(1) we get $\beta \lozenge \rho = \mathsf{r} :: (\omega'_{\cong}, \mathsf{q}!\lambda \cdot \eta)$ if $\mathsf{r} = \mathsf{p}$ and $\beta \lozenge \rho = \mathsf{r} :: (\omega'_{\cong}, \eta)$ if $\mathsf{r} \neq \mathsf{p}$. By Definition 4.3(1) $\mathsf{q}!\lambda \cdot \eta \in \mathcal{PE}(\bigoplus_{i\in I} \mathsf{q}!\lambda_i; P_i)$ and $\eta \in \mathcal{PE}(R_\mathsf{r})$ if $\mathsf{r} \neq \mathsf{p}$. We conclude that $\beta \lozenge \rho \in \mathcal{NE}(\mathsf{N} \parallel \mathcal{M})$ by Definition 5.8(1).
Let $\beta = \mathsf{pq}?\lambda$. By Definition 5.8(1) we have $\eta \in \mathcal{PE}(Q_k)$ if $\mathsf{r} = \mathsf{q}$ and $\eta \in \mathcal{PE}(R_\mathsf{r})$ if $\mathsf{r} \neq \mathsf{q}$. By Definition 8.1(1) we have $\beta \lozenge \rho = \mathsf{r} :: (\omega'_{\cong}, \mathsf{p}?\lambda \cdot \eta)$ if $\mathsf{r} = \mathsf{q}$ and $\beta \lozenge \rho = \mathsf{r} :: (\omega'_{\cong}, \eta)$ if $\mathsf{r} \neq \mathsf{q}$. By Definition 4.3(1) $\mathsf{p}?\lambda \cdot \eta \in \mathcal{PE}(\sum_{i\in I} \mathsf{p}?\lambda_i; Q_i)$ and $\eta \in \mathcal{PE}(R_\mathsf{r})$ if $\mathsf{r} \neq \mathsf{p}$. We conclude that $\beta \lozenge \rho \in \mathcal{NE}(\mathsf{N} \parallel \mathcal{M})$ by Definition 5.8(1).

(2) Assume now $\rho = \mathsf{r} :: (\omega_{\cong}, \eta) \in \mathcal{NE}(\mathsf{N} \parallel \mathcal{M})$. By Definition 5.8(1) we have $\omega = \mathsf{otr}(\mathcal{M})$. Let $\omega' = \mathsf{otr}(\mathcal{M}') \cong \beta \blacktriangleright \omega$.
Let $\beta = \mathsf{pq}!\lambda$. If $\mathsf{r} = \mathsf{p}$ and $\beta \blacklozenge \rho$ is defined, then we get $\beta \blacklozenge \rho = \mathsf{r} :: (\omega'_{\cong}, \eta')$ where $\eta = \mathsf{q}!\lambda \cdot \eta'$ by Definition 8.1(2). By Definition 5.8(1) we have $\mathsf{q}!\lambda \cdot \eta' \in \mathcal{PE}(\bigoplus_{i\in I} \mathsf{q}!\lambda_i; P_i)$. By Definition 4.3(1) $\eta' \in \mathcal{PE}(P_k)$. If $\mathsf{r} \neq \mathsf{p}$, by Definition 8.1(2) we have $\beta \blacklozenge \rho = \mathsf{r} :: (\omega'_{\cong}, \eta)$. By Definition 5.8(1) $\eta \in \mathcal{PE}(R_\mathsf{r})$. We conclude that $\beta \blacklozenge \rho \in \mathcal{NE}(\mathsf{N}' \parallel \mathcal{M}')$ by Definition 5.8(1).

Let $\beta = \mathsf{pq}?\lambda$. If $\mathsf{r} = \mathsf{q}$ and $\beta \blacklozenge \rho$ is defined, then by Definition 8.1(2) we get $\beta \blacklozenge \rho = \mathsf{r} :: (\omega'_{\cong}, \eta')$ where $\eta = \mathsf{p}?\lambda \cdot \eta'$. By Definition 5.8(1) we have $\mathsf{p}?\lambda \cdot \eta' \in \mathcal{PE}(\sum_{i\in I}\mathsf{p}?\lambda_i; Q_i)$ and $\eta' \in \mathcal{PE}(Q_k)$. If $\mathsf{r} \neq \mathsf{p}$, by Definition 8.1(2) we have $\beta \blacklozenge \rho = \mathsf{r} :: (\omega'_{\cong}, \eta)$. By Definition 5.8(1) $\eta \in \mathcal{PE}(R_{\mathsf{r}})$. We conclude that $\beta \blacklozenge \rho \in \mathcal{NE}(\mathsf{N}' \parallel \mathcal{M}')$ by Definition 5.8(1). □

We now define the total function nec, which yields sequences of n-events starting from pairs of o-traces and traces. We use the projection given in Definition 5.4(1).

Definition 8.8 (n-events from pairs of o-traces and traces)**.** *We define the* sequence of n-events corresponding to ω and τ *by*

$$\mathsf{nec}(\omega, \tau) = \rho_1; \cdots ; \rho_n$$

where

$$\rho_i = \mathsf{p}_i :: (\omega_{\cong}, \eta_i) \;\; \textit{if} \;\; \{\mathsf{p}_i\} = \mathsf{play}(\tau[i]) \textit{ and } \eta_i = \tau[1 \ldots i] @ \mathsf{p}_i$$

It is immediate to see that, if $\tau = \mathsf{pq}!\lambda$ or $\tau = \mathsf{pq}?\lambda$, then $\mathsf{nec}(\omega, \tau)$ consists only of the n-event $\mathsf{p} :: (\omega_{\cong}, \mathsf{q}!\lambda)$ or of the n-event $\mathsf{q} :: (\omega_{\cong}, \mathsf{p}?\lambda)$, respectively, because $\tau[1 \ldots 1] = \tau[1]$.

We show now that two n-events appearing in the sequence generated from a given pair (ω, τ) cannot be in conflict. Moreover, from $\mathsf{nec}(\omega, \tau)$ we can recover τ by means of the function $\mathsf{i/o}$ of Definition 5.3(2).

Lemma 8.9 . *Let* $\mathsf{nec}(\omega, \tau) = \rho_1; \cdots ; \rho_n$.

(1) *If* $1 \leq k, l \leq n$, *then* $\neg(\rho_k \# \rho_l)$;

(2) $\tau[i] = \mathsf{i/o}(\rho_i)$ *for all* i, $1 \leq i \leq n$.

Proof. (1) Let $\rho_i = \mathsf{p}_i :: (\omega_{\cong}, \eta_i)$ for all i, $1 \leq i \leq n$. If $\mathsf{p}_k \neq \mathsf{p}_l$, then ρ_k and ρ_l cannot be in conflict. If $\mathsf{p}_k = \mathsf{p}_l$, then by Definition 8.8 either $\eta_k < \eta_l$ or $\eta_k < \eta_l$. So in all cases we have $\neg(\rho_k \# \rho_l)$.

(2) Immediate from Definition 8.8. □

The following lemma relates the operators $\lozenge$ and $\blacklozenge$ with the mapping nec. This will be handy for the proof of Theorem 8.11.

Lemma 8.10 . (1) *Let* $\tau = \beta \cdot \tau'$ *and* $\omega = \beta \triangleright \omega'$. *If* $\mathsf{nec}(\omega, \tau) = \rho_1; \cdots ; \rho_n$ *and* $\mathsf{nec}(\omega', \tau') = \rho'_2; \cdots ; \rho'_n$, *then* $\beta \lozenge \rho'_i = \rho_i$ *for all* i, $2 \leq i \leq n$.

(2) *Let* $\tau = \beta \cdot \tau'$ *and* $\omega' = \beta \blacktriangleright \omega$. *If* $\mathsf{nec}(\omega, \tau) = \rho_1; \cdots ; \rho_n$ *and* $\mathsf{nec}(\omega', \tau') = \rho'_2; \cdots ; \rho'_n$, *then* $\beta \blacklozenge \rho_i = \rho'_i$ *for all* i, $2 \leq i \leq n$.

Proof. (1) Let $\rho_i = \mathsf{p}_i :: (\omega_{\cong}, \eta_i)$ for all i, $1 \leq i \leq n$ and $\rho'_i = \mathsf{p}_i :: (\omega'_{\cong}, \eta'_i)$ for all i, $2 \leq i \leq n$. Note that $\tau[i] = \tau'[i{-}1]$ for all i, $2 \leq i \leq n$. By Definition 8.8 $\eta_i = \tau[1 \ldots i] @ \mathsf{p}_i = (\beta \cdot \tau'[1 \ldots i{-}1]) @ \mathsf{p}_i$ for all i, $1 \leq i \leq n$ and $\eta'_i = \tau'[1 \ldots i-1] @ \mathsf{p}_i$ for all i, $2 \leq i \leq n$. Then by Lemma 8.4(1) we have $\beta \lozenge \rho'_i = \mathsf{p}_i :: ((\beta \triangleright \omega')_{\cong}, \beta @ \mathsf{p}_i \cdot \eta'_i) = \mathsf{p}_i :: (\omega_{\cong}, \eta_i) = \rho_i$ for all i, $2 \leq i \leq n$.

(2) From Fact (1) and Lemma 8.5(1). □

We end this subsection with the two theorems for networks discussed at the beginning of the whole section.

Theorem 8.11 . *If* $\mathsf{N} \parallel \mathcal{M} \xrightarrow{\tau} \mathsf{N}' \parallel \mathcal{M}'$, *then* $\mathsf{nec}(\mathsf{otr}(\mathcal{M}), \tau)$ *is a proving sequence in* $\mathcal{S}^{\mathcal{N}}(\mathsf{N} \parallel \mathcal{M})$.

Proof. The proof is by induction on τ. Let $\omega = \mathsf{otr}(\mathcal{M})$.

Case $\tau = \beta$. Assume first that $\beta = \mathsf{pq}!\lambda$. From $\mathsf{N} \parallel \mathcal{M} \xrightarrow{\beta} \mathsf{N}' \parallel \mathcal{M}'$ we get $\mathsf{p}[\![\bigoplus_{i\in I} \mathsf{q}!\lambda_i; P_i]\!] \in \mathsf{N}$ with $\lambda = \lambda_k$ for some $k \in I$. Thus $\mathsf{p}[\![P_k]\!] \in \mathsf{N}'$ and $\mathcal{M}' \equiv \mathcal{M} \cdot \langle \mathsf{p}, \lambda, \mathsf{q} \rangle$. By Definition 4.3(1) $\mathsf{q}!\lambda \in \mathcal{PE}(\bigoplus_{i\in I} \mathsf{q}!\lambda_i; P_i)$. By Definition 5.8(1) $\mathsf{p} :: (\omega_{\cong}, \mathsf{q}!\lambda) \in \mathcal{NE}(\mathsf{N} \parallel \mathcal{M})$. By Definition 8.8

$\mathsf{nec}(\omega,\beta) = \rho_1 = \mathsf{p} :: (\omega_{\cong}, \mathsf{q}!\lambda)$. Clearly, ρ_1 is a proving sequence in $\mathcal{S}^{\mathcal{N}}(\mathsf{N} \parallel \mathcal{M})$, since $\rho \prec \rho_1$ would imply $\rho = \mathsf{p} :: (\omega_{\cong}, \eta)$ for some η such that $\eta < \mathsf{q}!\lambda$, which is not possible.
Assume now that $\beta = \mathsf{pq}?\lambda$. In this case we get $\mathsf{q}[\![\sum_{i\in I} \mathsf{p}?\lambda_i; Q_i]\!] \in \mathsf{N}$ with $\lambda = \lambda_k$ for some $k \in I$. Thus $\mathsf{q}[\![Q_k]\!] \in \mathsf{N}'$ and $\mathcal{M} = \langle \mathsf{p}, \lambda, \mathsf{q}\rangle \cdot \mathcal{M}'$. With a similar reasoning as in the previous case, we obtain $\mathsf{nec}(\omega,\beta) = \rho_1 = \mathsf{q} :: (\omega_{\cong}, \mathsf{p}?\lambda)$. Since $\omega \cong \mathsf{pq}!\lambda \cdot \omega'$, where $\omega' = \mathsf{otr}(\mathcal{M}')$, it is immediate to see that ρ_1 is queue-justified. This implies that there is no event ρ in $\mathcal{NE}(\mathsf{N} \parallel \mathcal{M})$ such that $\rho \prec \rho_1$, and thus ρ_1 is a proving sequence in $\mathcal{S}^{\mathcal{N}}(\mathsf{N} \parallel \mathcal{M})$.
Case $\tau = \beta \cdot \tau'$ with $\tau' \neq \epsilon$. In this case, from $\mathsf{N} \parallel \mathcal{M} \xrightarrow{\tau} \mathsf{N}' \parallel \mathcal{M}'$ we get

$$\mathsf{N} \parallel \mathcal{M} \xrightarrow{\beta} \mathsf{N}'' \parallel \mathcal{M}'' \xrightarrow{\tau'} \mathsf{N}' \parallel \mathcal{M}'$$

for some $\mathsf{N}'', \mathcal{M}''$. Let $\omega' = \mathsf{otr}(\mathcal{M}'')$. By Lemma 8.7 $\omega = \beta \triangleright \omega'$. Let $\mathsf{nec}(\omega,\tau) = \rho_1; \cdots; \rho_n$ and $\mathsf{nec}(\omega',\tau') = \rho'_2; \cdots; \rho'_n$. By induction $\mathsf{nec}(\omega',\tau')$ is a proving sequence in $\mathcal{S}^{\mathcal{N}}(\mathsf{N}'' \parallel \mathcal{M}'')$. By Lemma 8.10(1) $\beta \lozenge \rho'_j = \rho_j$ for all j, $2 \leq j \leq n$. By Lemma 8.7(1) this implies $\rho_j \in \mathcal{NE}(\mathsf{N} \parallel \mathcal{M})$ for all j, $2 \leq j \leq n$. From the proof of the base case we know that $\rho_1 = \mathsf{p} :: (\omega_{\cong}, \beta @ \mathsf{p}) \in \mathcal{NE}(\mathsf{N} \parallel \mathcal{M})$ where $\{\mathsf{p}\} = \mathsf{play}(\beta)$. What is left to show is that $\rho_1; \cdots; \rho_n$ is a proving sequence in $\mathcal{S}^{\mathcal{N}}(\mathsf{N} \parallel \mathcal{M})$. By Lemma 8.9(1) no two events in this sequence can be in conflict. Let $\rho \in \mathcal{NE}(\mathsf{N} \parallel \mathcal{M})$ and $\rho \prec \rho_h$ for some h, $1 \leq h \leq n$. As argued in the base case, this implies $h > 1$. We distinguish two cases, depending on whether $\beta \blacklozenge \rho$ is defined or not.
If $\beta \blacklozenge \rho$ is defined, by Lemma 8.7(2) $\beta \blacklozenge \rho \in \mathcal{NE}(\mathsf{N}'' \parallel \mathcal{M}'')$ and by Lemma 8.6(2) we have $\beta \blacklozenge \rho \prec \beta \blacklozenge \rho_h$. Let $\rho' = \beta \blacklozenge \rho$. By Lemma 8.10(2) $\beta \blacklozenge \rho_j = \rho'_j$ for all $j, 2 \leq j \leq n$. Thus we have $\rho' \prec \rho'_h$. Since $\mathsf{nec}(\omega',\tau')$ is a proving sequence in $\mathcal{S}^{\mathcal{N}}(\mathsf{N}'' \parallel \mathcal{M}'')$, by Definition 3.6 there is $l < h$ such that either $\rho' = \rho'_l$ or $\rho' \# \rho'_l \prec \rho'_h$. In the first case we have $\rho = \beta \lozenge \rho' = \beta \lozenge \rho'_l = \rho_l$. In the second case, from $\rho' \# \rho'_l$ we deduce $\rho \# \rho_l$ by Lemma 8.6(3), and from $\rho'_l \prec \rho'_h$ we deduce $\rho_l \prec \rho_h$ by Lemma 8.6(1).
If $\beta \blacklozenge \rho$ is undefined, then by Definition 8.1(2) either $\rho = \rho_1$ or $\rho = \mathsf{p} :: (\omega_{\cong}, \pi \cdot \zeta)$ with $\pi \neq \beta @ \mathsf{p}$, which implies $\rho \# \rho_1$. In the first case we are done. So, suppose $\rho \# \rho_1$. Let $\pi' = \beta @ \mathsf{p}$. Since ρ and ρ_1 are events in $\mathcal{NE}(\mathsf{N} \parallel \mathcal{M})$, we may assume $\pi = \mathsf{q}!\lambda$ and $\pi' = \mathsf{q}!\lambda'$ and therefore $\beta = \mathsf{pq}!\lambda'$. Indeed, we know that $\mathsf{play}(\beta) = \mathsf{p}$, and β cannot be an input $\mathsf{qp}?\lambda'$ since in this case there should be $\rho_0 = \mathsf{p} :: (\omega_{\cong}, \mathsf{q}?\lambda) \in \mathcal{NE}(\mathsf{N} \parallel \mathcal{M})$ by narrowing, and the two input n-events ρ_0 and $\rho_1 = \mathsf{p} :: (\omega_{\cong}, \mathsf{q}?\lambda')$ could not be both justified by the same queue ω. Note that ρ cannot be a local cause of ρ_h, because $\rho_h = \mathsf{p} :: (\omega_{\cong}, \pi \cdot \zeta \cdot \eta)$ would imply $\rho_h \# \rho_1$, contradicting what said above. Therefore ρ is a cross-cause of ρ_h, so $\rho = \mathsf{p} :: (\omega_{\cong}, \pi \cdot \zeta' \cdot \mathsf{r}!\lambda'')$ and $\rho_h = \mathsf{r} :: (\omega_{\cong}, \zeta'' \cdot \mathsf{p}?\lambda'')$. We know that $\rho_h = \beta \lozenge \rho'_h$. By Definition 8.1(1) we have $\rho'_h = \mathsf{r} :: ((\omega \cdot \beta)_{\cong}, \zeta'' \cdot \mathsf{p}?\lambda'')$, because r is the receiver of a message sent by p and thus by construction $\mathsf{r} \neq \mathsf{p} = \mathsf{play}(\beta)$. Since ρ'_h is an input n-event in $\mathcal{NE}(\mathsf{N}'' \parallel \mathcal{M}'')$, it must either be justified by the queue $\omega \cdot \beta$ or have a cross-cause in $\mathcal{NE}(\mathsf{N}'' \parallel \mathcal{M}'')$. Since ρ_h is not justified by the queue ω (because $\rho \prec \rho_h$), the only way for ρ'_h to be justified by $\omega \cdot \beta$ would be that $\mathsf{pr}!\lambda'' = \beta$, that is $\mathsf{r} = \mathsf{q}$ and $\lambda'' = \lambda'$, and that (*) $(\omega @ \mathsf{p}) \restriction \mathsf{q} \lessapprox \bowtie \gtrapprox \zeta_0 \restriction \mathsf{p}$ and $(\zeta'' \cdot \mathsf{p}?\lambda') \restriction \mathsf{p} \lessapprox (\zeta_0 \cdot \mathsf{p}?\lambda' \cdot \chi) \restriction \mathsf{p}$ for some ζ_0 and χ, see Definition 5.7. This means that $\zeta_0 \restriction \mathsf{p}$ is the subsequence of $\zeta'' \restriction \mathsf{p}$ obtained by keeping all and only its inputs. Now, if $\rho'_h = \mathsf{q} :: ((\omega \cdot \beta)_{\cong}, \zeta'' \cdot \mathsf{p}?\lambda')$, then $\rho_h = \mathsf{q} :: (\omega_{\cong}, \zeta'' \cdot \mathsf{p}?\lambda')$. Since $\rho = \mathsf{p} :: (\omega_{\cong}, \mathsf{q}!\lambda \cdot \zeta' \cdot \mathsf{q}!\lambda')$ is a cross-cause of ρ_h, we have (**) $(\omega @ \mathsf{p} \cdot \mathsf{q}!\lambda \cdot \zeta') \restriction \mathsf{q} \lessapprox \bowtie \gtrapprox (\omega @ \mathsf{q} \cdot \zeta_1 \cdot \chi') \restriction \mathsf{p}$ and $(\zeta'' \cdot \mathsf{p}?\lambda') \restriction \mathsf{p} \lessapprox (\zeta_1 \cdot \mathsf{p}?\lambda' \cdot \chi') \restriction \mathsf{p}$ for some ζ_1 and χ', see Definition 5.6(1)(b). It follows that the inputs in $\zeta_1 \restriction \mathsf{p}$ coincide with the inputs in $\zeta'' \restriction \mathsf{p}$ and thus with those in $\zeta_0 \restriction \mathsf{p}$. From (*) we know that all inputs in $\zeta_0 \restriction \mathsf{p}$ match some output in $(\omega @ \mathsf{p}) \restriction \mathsf{q}$. Therefore no

input in $(\omega @ \mathsf{q} \cdot \zeta_1 \cdot \chi') \pitchfork \mathsf{p}$ can match the output $\mathsf{q}!\lambda$ in $(\omega @ \mathsf{p} \cdot \mathsf{q}!\lambda \cdot \zeta') \pitchfork \mathsf{q}$, contradicting (**). Hence ρ'_h must have a cross-cause in $\mathcal{NE}(\mathsf{N}'' \parallel \mathcal{M}'')$. Let ρ' be such a cross-cause. Then $\rho' = \mathsf{p} :: ((\omega \cdot \beta)_{\cong}, \zeta_2 \cdot \mathsf{r}!\lambda'')$. Since $\mathsf{nec}(\omega', \tau')$ is a proving sequence in $\mathcal{S}^{\mathcal{N}}(\mathsf{N}'' \parallel \mathcal{M}'')$, by Definition 3.6 there is $l < h$ such that either $\rho' = \rho'_l$ or $\rho' \# \rho'_l \prec \rho'_h$. In the first case $\beta \lozenge \rho' = \beta \lozenge \rho'_l = \rho_l \prec \rho_h$, and $(\beta \lozenge \rho') \# \rho$ because $\beta \lozenge \rho' = \mathsf{p} :: (\omega_{\cong}, \pi' \cdot \zeta_2 \cdot \mathsf{r}!\lambda'')$. In the second case, from $\rho' \# \rho'_l \prec \rho'_h$ we derive $\beta \lozenge \rho' \# \beta \lozenge \rho'_l \prec \beta \lozenge \rho'_h$ by Lemma 8.6(3) and (1). This implies $\rho_l = \beta \lozenge \rho'_l = \mathsf{p} :: (\omega_{\cong}, \pi' \cdot \zeta'_2 \cdot \mathsf{r}!\lambda'')$. Hence $\rho \# \rho_l \prec \rho_h$. □

Theorem 8.12 . *If $\rho_1; \cdots; \rho_n$ is a proving sequence in $\mathcal{S}^{\mathcal{N}}(\mathsf{N} \parallel \mathcal{M})$, then $\mathsf{N} \parallel \mathcal{M} \xrightarrow{\tau} \mathsf{N}' \parallel \mathcal{M}'$ where $\tau = \mathsf{i/o}(\rho_1) \cdots \mathsf{i/o}(\rho_n)$.*

Proof. The proof is by induction on the length n of the proving sequence. Let $\omega = \mathsf{otr}(\mathcal{M})$.
Case $n = 1$. Let $\mathsf{i/o}(\rho_1) = \beta$ where $\beta = \mathsf{pq}?\lambda$. The proof for $\beta = \mathsf{pq}!\lambda$ is similar and simpler. By Definition 5.8(1) $\rho_1 = \mathsf{q} :: (\omega_{\cong}, \zeta \cdot \mathsf{p}?\lambda)$. Note that it must be $\zeta = \epsilon$, since otherwise we would have $\mathsf{q} :: (\omega_{\cong}, \zeta) \in \mathcal{NE}(\mathsf{N} \parallel \mathcal{M})$ by narrowing, where $\mathsf{q} :: (\omega_{\cong}, \zeta) \prec \rho_1$ by Definition 5.6(1)(a), contradicting the hypothesis that ρ_1 is minimal. Moreover, ρ_1 cannot be justified by an output n-event $\rho \in \mathcal{NE}(\mathsf{N} \parallel \mathcal{M})$, because this would imply $\rho \prec \rho_1$, contradicting again the minimality of ρ_1. Hence, by Definition 5.8(1) $\rho_1 = \mathsf{q} :: (\omega_{\cong}, \mathsf{p}?\lambda)$ must be queue-justified, which means that $\omega \cong \mathsf{pq}!\lambda \cdot \omega'$. Thus $\mathcal{M} \equiv \langle \mathsf{p}, \lambda, \mathsf{q} \rangle \cdot \mathcal{M}'$, where $\mathsf{otr}(\mathcal{M}') = \omega'$. By Definition 4.3(1) and Definition 5.8(1) we have $\mathsf{N} \equiv \mathsf{q}[\![\Sigma_{i \in I} \mathsf{p}?\lambda_i; Q_i]\!] \parallel \mathsf{N}_0$ where $\lambda_k = \lambda$ for some $k \in I$. We may then conclude that $\mathsf{N} \parallel \mathcal{M} \xrightarrow{\beta} \mathsf{q}[\![Q_k]\!] \parallel \mathsf{N}_0 \parallel \mathcal{M}' = \mathsf{N}' \parallel \mathcal{M}'$.

Case $n > 1$. Let $\mathsf{i/o}(\rho_1) = \beta$ and $\mathsf{N} \parallel \mathcal{M} \xrightarrow{\beta} \mathsf{N}'' \parallel \mathcal{M}''$ be the corresponding transition as obtained from the base case. We show that $\beta \blacklozenge \rho_j$ is defined for all j, $2 \le j \le n$. If $\beta \blacklozenge \rho_k$ were undefined for some k, $2 \le k \le n$, then by Definition 8.1(2) either $\rho_k = \rho_1$ or $\rho_k = \mathsf{p} :: (\omega_{\cong}, \pi \cdot \zeta)$ with $\pi \neq \beta @ \mathsf{p}$, which implies $\rho_k \# \rho_1$. So both cases are impossible. Thus, by Lemma 8.7(2) we may define $\rho'_j = \beta \blacklozenge \rho_j \in \mathcal{NE}(\mathsf{N}'' \parallel \mathcal{M}'')$ for all j, $2 \le j \le n$. We show that $\rho'_2; \cdots; \rho'_n$ is a proving sequence in $\mathcal{S}^{\mathcal{N}}(\mathsf{N}'' \parallel \mathcal{M}'')$. By Lemma 8.5(2) $\rho_j = \beta \lozenge \rho'_j$ for all j, $2 \le j \le n$. Then by Lemma 8.6(3) no two n-events in the sequence $\rho'_2; \cdots; \rho'_n$ can be in conflict.
Let $\rho \in \mathcal{NE}(\mathsf{N}'' \parallel \mathcal{M}'')$ and $\rho \prec \rho'_h$ for some h, $2 \le h \le n$. By Lemma 8.7(1) $\beta \lozenge \rho \in \mathcal{NE}(\mathsf{N} \parallel \mathcal{M})$ and then by Lemma 8.6(1) $\beta \lozenge \rho \prec \beta \lozenge \rho'_h = \rho_h$. Let $\rho' = \beta \lozenge \rho$. Therefore $\rho' \prec \rho_h$. Since $\rho_1; \cdots; \rho_n$ is a proving sequence in $\mathcal{S}^{\mathcal{N}}(\mathsf{N} \parallel \mathcal{M})$, by Definition 3.6 there is $l < h$ such that either $\rho' = \rho_l$ or $\rho' \# \rho_l \prec \rho_h$. In the first case, by Lemma 8.5(1) we get $\rho = \beta \blacklozenge \rho' = \beta \blacklozenge \rho_l = \rho'_l$. In the second case, by Lemma 8.6(2) and (4) we get $\rho \# \rho'_l \prec \rho'_h$.
We have shown that $\rho'_2; \cdots; \rho'_n$ is a proving sequence in $\mathcal{S}^{\mathcal{N}}(\mathsf{N}'' \parallel \mathcal{M}'')$. By induction $\mathsf{N}'' \parallel \mathcal{M}'' \xrightarrow{\tau'} \mathsf{N}' \parallel \mathcal{M}'$ where $\tau' = \mathsf{i/o}(\rho'_2) \cdots \mathsf{i/o}(\rho'_n)$. Let $\tau = \mathsf{i/o}(\rho_1) \cdot \ldots \cdot \mathsf{i/o}(\rho_n)$. Since $\mathsf{i/o}(\rho'_j) = \mathsf{i/o}(\rho_j)$ for all $j, 2 \le j \le n$, we have $\tau = \beta \cdot \tau'$. Hence $\mathsf{N} \parallel \mathcal{M} \xrightarrow{\beta} \mathsf{N}'' \parallel \mathcal{M}'' \xrightarrow{\tau'} \mathsf{N}' \parallel \mathcal{M}'$ is the required transition sequence. □

8.2. **Transition Sequences of Global Types and Proving Sequences of their ESs.**

We introduce two operators $\circ$ and $\bullet$ for global events, which play the same role as the operators $\lozenge$ and $\blacklozenge$ for network events. In defining these operators we must make sure that, in the resulting g-event $[\omega', \tau']_{\sim}$, the trace τ' is ω'-pointed, see Definition 7.12(1) and (2).

Let us start with the formal definition, and then we shall explain it in detail.

Definition 8.13 (Retrieval and residual of a g-event with respect to a communication)**.**

(1) *(Retrieval of a g-event before a communication) The operator $\circ$ applied to a communication β and a g-event δ is defined by cases on β:*

$$\mathsf{pq}!\lambda \circ [\omega \cdot \mathsf{pq}!\lambda, \tau]_\sim = \begin{cases} [\omega, \mathsf{pq}!\lambda \cdot \tau]_\sim & \text{if } \mathsf{pq}!\lambda \cdot \tau \text{ is } \omega\text{-pointed} \\ [\omega, \tau]_\sim & \text{otherwise} \end{cases}$$

$$\mathsf{pq}?\lambda \circ [\omega, \tau]_\sim = \begin{cases} [\mathsf{pq}!\lambda \cdot \omega, \mathsf{pq}?\lambda \cdot \tau]_\sim & \text{if } \mathsf{q} \in \mathsf{play}(\tau) \\ [\mathsf{pq}!\lambda \cdot \omega, \tau]_\sim & \text{if } \mathsf{q} \notin \mathsf{play}(\tau) \end{cases}$$

(2) *(Residual of a g-event after a communication) The operator $\bullet$ applied to a communication β and a g-event δ is defined by cases on β:*

$$\mathsf{pq}!\lambda \bullet [\omega, \tau]_\sim = \begin{cases} [\omega \cdot \mathsf{pq}!\lambda, \tau']_\sim & \text{if } \tau \approx_\omega \mathsf{pq}!\lambda \cdot \tau' \text{ with } \tau' \neq \epsilon \\ [\omega \cdot \mathsf{pq}!\lambda, \tau]_\sim & \text{if } \mathsf{p} \notin \mathsf{play}(\tau) \end{cases}$$

$$\mathsf{pq}?\lambda \bullet [\mathsf{pq}!\lambda \cdot \omega, \tau]_\sim = \begin{cases} [\omega, \tau']_\sim & \text{if } \tau \approx_{\mathsf{pq}!\lambda \cdot \omega} \mathsf{pq}?\lambda \cdot \tau' \text{ and } \tau' \neq \epsilon \\ [\omega, \tau]_\sim & \text{if } \mathsf{q} \notin \mathsf{play}(\tau) \end{cases}$$

Note that the operators $\circ$ and $\bullet$ preserve the communication of g-events, namely $\mathsf{i/o}(\beta \circ \delta) = \mathsf{i/o}(\beta \bullet \delta) = \mathsf{i/o}(\delta)$. It is also immediate to see that they transform the o-trace as the operators $\Diamond$ and $\blacklozenge$, see Definition 8.2. So we will explain only the transformation of the trace τ.

Consider first the case of $\mathsf{pq}?\lambda \circ [\omega, \tau]_\sim$, which is the simplest one. To obtain the retrieval of $[\omega, \tau]_\sim$ before $\mathsf{pq}?\lambda$, we start by prefixing the trace τ by the input $\mathsf{pq}?\lambda$. The resulting trace will be ω-pointed if and only if the added input is a local cause of some subsequent communication, and this holds precisely when $\mathsf{q} \in \mathsf{play}(\tau)$. If this is not the case, then we remove the new input $\mathsf{pq}?\lambda$ from the trace. Consider now the case of $\mathsf{pq}!\lambda \circ [\omega \cdot \mathsf{pq}!\lambda, \tau]_\sim$. Here we extend the trace as expected, but checking the pointedness condition is more involved, since we need to check that the added output is either a local cause or a cross-cause of some communication in τ. Note that we do not need to check that all the other outputs in τ still satisfy the 2nd condition of pointedness, since, letting $\omega' = \omega \cdot \mathsf{pq}!\lambda$ and $\tau' = \mathsf{pq}!\lambda \cdot \tau$, for any output $\mathsf{rs}!\lambda$ occurring in τ we have that the number of outputs from r to s preceding it in $\omega \cdot \tau'$ is equal to the number of outputs from r to s preceding it in $\omega' \cdot \tau$. Therefore we could have replaced the condition "if $\mathsf{pq}!\lambda \cdot \tau$ is ω-pointed" by the more detailed (but equivalent) condition "if $\mathsf{p} \in \mathsf{play}(\tau)$ or $1 + |\omega| \propto^{\omega \cdot \mathsf{pq}!\lambda \cdot \tau} j + |\omega|$ for some $j > 1$", which only concerns the new output. If this condition is not satisfied, then the new output is removed from the trace. For instance, for the g-event $[\mathsf{pq}!\lambda, \mathsf{pq}?\lambda]_\sim$, where $\omega = \mathsf{pq}!\lambda$ and $\tau = \mathsf{pq}?\lambda$, we have $\mathsf{p} \notin \mathsf{play}(\tau)$, but $1 \propto^{\mathsf{pq}!\lambda \cdot \mathsf{pq}?\lambda} 2$, thus $\mathsf{pq}!\lambda \circ [\mathsf{pq}!\lambda, \mathsf{pq}?\lambda]_\sim = [\epsilon, \mathsf{pq}!\lambda \cdot \mathsf{pq}?\lambda]_\sim$. On the other hand, for the g-event $[\mathsf{pq}!\lambda, \mathsf{rs}!\lambda' \cdot \mathsf{rs}?\lambda']_\sim$, where $\omega = \mathsf{pq}!\lambda$ and $\tau = \mathsf{rs}!\lambda' \cdot \mathsf{rs}?\lambda'$, we have $\mathsf{p} \notin \mathsf{play}(\tau)$ and $\neg(1 \propto^{\mathsf{pq}!\lambda \cdot \mathsf{rs}!\lambda' \cdot \mathsf{rs}?\lambda'} 2)$ and $\neg(1 \propto^{\mathsf{pq}!\lambda \cdot \mathsf{rs}!\lambda' \cdot \mathsf{rs}?\lambda'} 3)$, so $\mathsf{pq}!\lambda \circ [\mathsf{pq}!\lambda, \mathsf{rs}!\lambda' \cdot \mathsf{rs}?\lambda']_\sim = [\epsilon, \mathsf{rs}!\lambda' \cdot \mathsf{rs}?\lambda']_\sim$. Note that $\mathsf{pq}!\lambda \circ [\omega, \tau]_\sim$ is undefined when the queue ω does not end with $\mathsf{pq}!\lambda$, while $\mathsf{pq}?\lambda \circ [\omega, \tau]_\sim$ is always defined.

Next, consider the definition of $\mathsf{pq}!\lambda \bullet [\omega, \tau]_\sim$. If the first communication with player p in τ is the output $\mathsf{pq}!\lambda$, then this output can be brought to the head of the trace using the equivalence $\approx_\omega$. In this case, we obtain the residual of $[\omega, \tau]_\sim$ after $\mathsf{pq}!\lambda$ by removing the message $\mathsf{pq}!\lambda$ from the head of the trace, provided this does not result in the empty trace (otherwise, the residual is undefined). Then, letting $\omega' = \omega \cdot \mathsf{pq}!\lambda$, it is easy to see that the trace τ' is ω'-pointed, since it is a suffix of $\tau = \mathsf{pq}!\lambda \cdot \tau'$ which is ω-pointed (see Lemma 7.9). On the other hand, if $\mathsf{p} \notin \mathsf{play}(\tau)$, then the residual of $[\omega, \tau]_\sim$ after $\mathsf{pq}!\lambda$ is

simply obtained by leaving the trace unchanged. In this case, letting again $\omega' = \omega \cdot \mathsf{pq}!\lambda$, the ω'-pointedness of τ follows immediately from its ω-pointedness, since τ contains no output from p to q and therefore the addition of the output $\mathsf{pq}!\lambda$ at the end of the queue does not affect the pointedness of τ. For instance, consider the g-event $[\mathsf{pr}!\lambda', \mathsf{pr}?\lambda']_\sim$ where $\omega = \mathsf{pr}!\lambda'$ and $\tau = \mathsf{pr}?\lambda'$. Observe that p occurs in τ, but $\mathsf{p} \notin \mathsf{play}(\tau)$. Then we have $\mathsf{pq}!\lambda \bullet [\mathsf{pr}!\lambda', \mathsf{pr}?\lambda']_\sim = [\mathsf{pr}!\lambda' \cdot \mathsf{pq}!\lambda, \mathsf{pr}?\lambda']_\sim$.
The case of $\mathsf{pq}?\lambda \bullet [\mathsf{pq}!\lambda \cdot \omega, \tau]_\sim$ is entirely similar.

It is easy to verify that the definitions of $\circ$ and $\bullet$ given in Definition 8.13 may be rewritten in the more concise form given in the following lemma, which is analogous to Lemma 8.4.

Lemma 8.14 . (1) *If $\beta \circ [\omega, \tau]_\sim$ is defined, then $(\beta \triangleright \omega) \cdot \beta \cdot \tau$ is well formed and*
$$\beta \circ [\omega, \tau]_\sim = [\beta \triangleright \omega, \beta \lceil_{(\beta \triangleright \omega)} \tau]_\sim$$
(2) *If $\beta \bullet [\omega, \beta \lceil_\omega \tau]_\sim$ is defined, then $\omega \cdot \beta \cdot \tau$ is well formed and*
$$\beta \bullet [\omega, \beta \lceil_\omega \tau]_\sim = [\beta \blacktriangleright \omega, \tau]_\sim$$

We shall now relate the operators $\circ$ and $\bullet$ with the function ev, which builds g-events, see Definition 7.14. To this end, we first prove the following lemma, which shows how the filtering of a trace gets affected when the trace is prefixed by a communication.

Lemma 8.15 . (1) *Let $\beta \triangleright \omega$ be defined and $\omega' = \beta \triangleright \omega$. Let τ, τ' be such that τ' is $(\omega' \cdot \beta \cdot \tau)$-pointed. Then*
$$(\beta \cdot \tau) \lceil_{\omega'} \tau' = \beta \lceil_{\omega'} (\tau \lceil_\omega \tau')$$
(2) *Let $\beta \blacktriangleright \omega$ be defined and $\omega' = \beta \blacktriangleright \omega$. Let τ, τ' be such that τ' is $(\omega \cdot \beta \cdot \tau)$-pointed. Then*
$$(\beta \cdot \tau) \lceil_\omega \tau' = \beta \lceil_\omega (\tau \lceil_{\omega'} \tau')$$

Proof. (1) We show $(\beta \cdot \tau) \lceil_{\omega'} \tau' = \beta \lceil_{\omega'} (\tau \lceil_\omega \tau')$ by induction on τ.
Case $\tau = \epsilon$. In this case both the LHS and RHS reduce to $\beta \lceil_{\omega'} \tau'$, for whatever ω.
Case $\tau = \tau'' \cdot \beta'$. By Definition 7.13 we obtain for the LHS:
$$(\beta \cdot \tau'' \cdot \beta') \lceil_{\omega'} \tau' = \begin{cases} (\beta \cdot \tau'') \lceil_{\omega'} (\beta' \cdot \tau') & \text{if } \beta' \cdot \tau' \text{ is } (\omega' \cdot \beta \cdot \tau'')\text{-pointed} \\ (\beta \cdot \tau'') \lceil_{\omega'} \tau' & \text{otherwise} \end{cases}$$
By Definition 7.13 (applied to the internal filtering) we obtain for the RHS:
$$\beta \lceil_{\omega'} ((\tau'' \cdot \beta') \lceil_\omega \tau') = \begin{cases} \beta \lceil_{\omega'} (\tau'' \lceil_\omega (\beta' \cdot \tau')) & \text{if } \beta' \cdot \tau' \text{ is } (\omega \cdot \tau'')\text{-pointed} \\ \beta \lceil_{\omega'} (\tau'' \lceil_\omega \tau') & \text{otherwise} \end{cases}$$
We distinguish two cases, according to whether β is an input or an output.
Suppose first that β is an output. Then $\omega = \omega' \cdot \beta$. The side condition, i.e. the requirement that $\beta' \cdot \tau'$ be $(\omega \cdot \tau'')$-pointed, is the same in both cases. We may then immediately conclude that LHS = RHS using the induction hypothesis.

Suppose now that β is an input. Then $\omega' = \overline{\beta} \cdot \omega$. Observe that, since $(\omega \cdot \tau'')$ is obtained from $(\omega' \cdot \beta \cdot \tau'') = (\overline{\beta} \cdot \omega \cdot \beta \cdot \tau'')$ by erasing a pair of matching communications, $(\beta' \cdot \tau')$ is $(\omega \cdot \tau'')$-pointed if and only if $(\beta' \cdot \tau')$ is $(\omega' \cdot \beta \cdot \tau'')$-pointed. Then we may again conclude by induction.

(2) follows from (1) since $\beta \triangleright (\beta \blacktriangleright \omega) = \omega$. □

We may now prove the following:

Lemma 8.16 . (1) *If $\beta \triangleright \omega$ is defined, then $\beta \circ \mathsf{ev}(\omega, \tau) = \mathsf{ev}(\beta \triangleright \omega, \beta \cdot \tau)$.*
(2) *If $\tau \neq \epsilon$ and $\beta \blacktriangleright \omega$ is defined, then $\beta \bullet \mathsf{ev}(\omega, \beta \cdot \tau) = \mathsf{ev}(\beta \blacktriangleright \omega, \tau)$.*

Proof. Definition 7.14 and Lemmas 8.14 and 8.15 with $\tau' = \epsilon$ imply (1) and (2) since:

$$\begin{array}{lllll}
(1) & \beta \circ \mathsf{ev}(\omega, \tau) & = & \beta \circ [\omega, \tau \lceil_\omega \epsilon]_\sim & \text{by Definition 7.14} \\
 & & = & [\omega', \beta \lceil_{\omega'} (\tau \lceil_\omega \epsilon)]_\sim & \text{by Lemma 8.14(1)} \\
 & & = & [\omega', (\beta \cdot \tau) \lceil_{\omega'} \epsilon]_\sim & \text{by Lemma 8.15(1)} \\
 & \mathsf{ev}(\omega', \beta \cdot \tau) & = & [\omega', (\beta \cdot \tau) \lceil_{\omega'} \epsilon]_\sim & \text{by Definition 7.14} \\
 & & & & \text{where } \omega' = \beta \triangleright \omega \\
(2) & \beta \bullet \mathsf{ev}(\omega, \beta \cdot \tau) & = & \beta \bullet [\omega, (\beta \cdot \tau) \lceil_\omega \epsilon]_\sim & \text{by Definition 7.14} \\
 & & = & \beta \bullet [\omega, \beta \lceil_\omega (\tau \lceil_{\omega'} \epsilon)]_\sim & \text{by Lemma 8.15(2)} \\
 & & = & [\omega', \tau \lceil_{\omega'} \epsilon]_\sim & \text{by Lemma 8.14(2)} \\
 & \mathsf{ev}(\omega', \tau) & = & [\omega', \tau \lceil_{\omega'} \epsilon]_\sim & \text{by Definition 7.14} \\
 & & & & \text{where } \omega' = \beta \blacktriangleright \omega
\end{array}$$

□

Lemma 8.17 is the analogous of Lemma 8.5 as regards the first two statements. The remaining two statements establish some commutativity properties of the mappings $\circ$ and $\bullet$ when applied to two communications with different players. These properties rely on the corresponding commutativity properties for the mappings $\triangleright$ and $\blacktriangleright$ on o-traces, given in Lemma 8.3. Note that these properties are needed for $\circ$ and $\bullet$ whereas they were not needed for $\Diamond$ and $\blacklozenge$, because the Rules [IComm-Out] and [IComm-In] of Figure 5 allow transitions to occur inside sgts, whereas the LTS for networks only allows transitions for top-level communications.

Lemma 8.17 . (1) *If $\beta \circ \delta$ is defined, then $\beta \bullet (\beta \circ \delta) = \delta$.*
(2) *If $\beta \bullet \delta$ is defined, then $\beta \circ (\beta \bullet \delta) = \delta$.*
(3) *If both $\beta_1 \circ \delta$, $\beta_2 \circ \delta$ are defined, and $\mathsf{play}(\beta_1) \cap \mathsf{play}(\beta_2) = \emptyset$, then $\beta_1 \circ (\beta_2 \circ \delta)$ is defined and $\beta_1 \circ (\beta_2 \circ \delta) = \beta_2 \circ (\beta_1 \circ \delta)$.*
(4) *If both $\beta_2 \bullet \delta$, $\beta_2 \bullet (\beta_1 \circ \delta)$ are defined, and $\mathsf{play}(\beta_1) \cap \mathsf{play}(\beta_2) = \emptyset$, then $\beta_1 \circ (\beta_2 \bullet \delta) = \beta_2 \bullet (\beta_1 \circ \delta)$.*

Proof. Statements (1) and (2) immediately follow from Lemma 8.14. In the proofs of the remaining statements we convene that "β is required in $\tau_1 \cdot \beta \cdot \tau_2$" is short for "the shown occurrence of β is required in $\tau_1 \cdot \beta \cdot \tau_2$" and similarly for "$\beta$ matches an input in $\tau_1 \cdot \beta \cdot \tau_2$".

(3) Let $\delta = [\omega, \tau]_\sim$. Since $\beta_i \circ \delta$ is defined for $i \in \{1, 2\}$, by Lemma 8.14(1) $\omega_i = \beta_i \triangleright \omega$ is defined for $i \in \{1, 2\}$. Then by Lemma 8.3(1) $\beta_1 \triangleright (\beta_2 \triangleright \omega) \cong \beta_2 \triangleright (\beta_1 \triangleright \omega)$. Let $\omega' = \beta_1 \triangleright (\beta_2 \triangleright \omega)$. Using Lemma 8.14(1) we get for $i \in \{1, 2\}$

$$\delta_i = \beta_i \circ [\omega, \tau]_\sim = [\omega_i, \beta_i \lceil_{\omega_i} \tau]_\sim$$

Using again Lemma 8.14(1) we get

$$\beta_2 \circ \delta_1 = \beta_2 \circ [\omega_1, \beta_1 \lceil_{\omega_1} \tau]_\sim = [\omega', \beta_2 \lceil_{\omega'} (\beta_1 \lceil_{\omega_1} \tau)]_\sim$$

Similarly

$$\beta_1 \circ \delta_2 = \beta_1 \circ [\omega_2, \beta_2 \lceil_{\omega_2} \tau]_\sim = [\omega', \beta_1 \lceil_{\omega'} (\beta_2 \lceil_{\omega_2} \tau)]_\sim$$

We want to prove that

$$(*) \quad \beta_1 \lceil_{\omega'} (\beta_2 \lceil_{\omega_2} \tau) \approx_{\omega'} \beta_2 \lceil_{\omega'} (\beta_1 \lceil_{\omega_1} \tau)$$

In the proof of (*) we will use the following facts, where $h, k \in \{1, 2\}$ and $h \neq k$:

(a) $\beta_h \cdot \beta_k \cdot \tau \approx_{\omega'} \beta_k \cdot \beta_h \cdot \tau$;
(b) if $\beta_h \cdot \tau$ is ω'-pointed and $\beta_k \cdot \tau$ is not ω_k-pointed, then $\beta_h \cdot \tau$ is ω_h-pointed;
(c) if $\beta_h \cdot \tau$ is ω_h-pointed and $\beta_k \cdot \tau$ is not ω_k-pointed, then $\beta_h \cdot \tau$ is ω'-pointed;
(d) $\beta_h \cdot \beta_k \cdot \tau$ is ω'-pointed iff $\beta_h \cdot \tau$ is ω_h-pointed and $\beta_k \cdot \tau$ is ω_k-pointed.

Fact (a). We show that $\beta_h \cdot \beta_k \cdot \tau$ ω'-swaps to $\beta_k \cdot \beta_h \cdot \tau$. By hypothesis $\mathsf{play}(\beta_h) \cap \mathsf{play}(\beta_k) = \emptyset$, so it is enough to show that β_k does not match β_h in the trace $\omega' \cdot \beta_h \cdot \beta_k \cdot \tau = (\beta_h \triangleright (\beta_k \triangleright \omega)) \cdot \beta_h \cdot \beta_k \cdot \tau$. Suppose that β_h is an output and β_k is an input such that $\overline{\beta_k} = \beta_h$. Since $\delta_h = \beta_h \circ \delta$ is defined and β_h is an output, it must be $\omega \cong \omega_h \cdot \beta_h$. Then, since $\delta_k = \beta_k \circ \delta$ is defined and β_k is an input and $\overline{\beta_k} = \beta_h$, we get $\beta_k \triangleright \omega = \overline{\beta}_k \cdot \omega \cong \overline{\beta}_k \cdot \omega_h \cdot \beta_h \cong \beta_h \cdot \omega_h \cdot \beta_h$. Then $\omega' = \beta_h \triangleright (\beta_k \triangleright \omega) \cong \beta_h \cdot \omega_h$. Clearly, β_k matches the initial output β_h in the trace $\omega' \cdot \beta_h \cdot \beta_k \cdot \tau$ since β_k is the first input in the trace and the initial β_h is the first complementary output in the trace. Therefore β_k does not match its adjacent output β_h.

Fact (b). If β_h is required in $\beta_h \cdot \tau$ - a condition that is always true when β_h is an input and $\beta_h \cdot \tau$ is ω'-pointed - then $\beta_h \cdot \tau$ is ω_0-pointed for all ω_0.
We may then assume that β_h is an output that is not required in $\beta_h \cdot \tau$.
If β_k is an output, then $\omega_h \cong \omega' \cdot \beta_k$. If β_h matches an input in $\omega' \cdot \beta_h \cdot \tau$, then β_h matches the same input in $\omega_h \cdot \beta_h \cdot \tau$, since $\mathsf{play}(\beta_h) \cap \mathsf{play}(\beta_k) = \emptyset$.
If β_k is an input, then $\omega' \cong \overline{\beta_k} \cdot \omega_h$. Suppose $\beta_h = \mathsf{pq}!\lambda$ and $\beta_k = \mathsf{rs}?\lambda'$. Observe that it must be $\mathsf{p} \neq \mathsf{r}$ or $\mathsf{q} \neq \mathsf{s}$, because otherwise no input $\mathsf{pq}?\lambda$ could occur in τ since $\beta_k \cdot \tau$ is not ω_k-pointed, contradicting the hypotheses that $\beta_h \cdot \tau$ is ω'-pointed and β_h is not required in $\beta_h \cdot \tau$. Then the presence of $\overline{\beta_k} = \mathsf{rs}!\lambda$ cannot affect the multiplicity of $\mathsf{pq}!$ or $\mathsf{pq}?$ in any trace. Therefore, if β_h matches an input in $\omega' \cdot \beta_h \cdot \tau$, then β_h matches the same input in $\omega_h \cdot \beta_h \cdot \tau$.

Fact (c). Again, we may assume that β_h is an output that is not required in $\beta_h \cdot \tau$.
If β_k is an output, then $\omega_h \cong \omega' \cdot \beta_k$. If β_h matches an input in $\omega_h \cdot \beta_h \cdot \tau$, then β_h matches the same input in $\omega' \cdot \beta_h \cdot \tau$, since $\mathsf{play}(\beta_h) \cap \mathsf{play}(\beta_k) = \emptyset$.
If β_k is an input, then $\omega' \cong \overline{\beta_k} \cdot \omega_h$. Let $\beta_h = \mathsf{pq}!\lambda$ and $\beta_k = \mathsf{rs}?\lambda'$. Again, it must be $\mathsf{p} \neq \mathsf{r}$ or $\mathsf{q} \neq \mathsf{s}$, because otherwise no input $\mathsf{pq}?\lambda$ could occur in τ since $\beta_k \cdot \tau$ is not ω_k-pointed, contradicting the hypotheses that $\beta_h \cdot \tau$ is ω_h-pointed and β_h is not required in $\beta_h \cdot \tau$. Therefore, if β_h matches an input in $\omega_h \cdot \beta_h \cdot \tau$, then β_h matches the same input in $\omega' \cdot \beta_h \cdot \tau$.

Fact (d). From $\mathsf{play}(\beta_h) \cap \mathsf{play}(\beta_k) = \emptyset$ it follows that β_h is required in $\beta_h \cdot \beta_k \cdot \tau$ iff β_h is required in $\beta_h \cdot \tau$, and similarly for β_k. Let us then assume that β_h and β_k are not both required in $\beta_h \cdot \beta_k \cdot \tau$, i.e., that at least one of them is an output not required in $\beta_h \cdot \beta_k \cdot \tau$.
If both β_h and β_k are outputs, then $\omega_h \cong \omega' \cdot \beta_k$. Then β_h matches an input in $\omega' \cdot \beta_h \cdot \beta_k \cdot \tau$ iff β_h matches the same input in $\omega_h \cdot \beta_h \cdot \tau$, since $\beta_h \cdot \beta_k \cdot \tau \approx_{\omega'} \beta_k \cdot \beta_h \cdot \tau$ by Fact (a).
Let $\beta_h = \mathsf{pq}!\lambda$ and $\beta_k = \mathsf{rs}?\lambda'$, where β_h is not required in $\beta_h \cdot \beta_k \cdot \tau$. Then $\omega' \cong \overline{\beta_k} \cdot \omega_h$. Therefore β_h matches an input in $\omega' \cdot \beta_h \cdot \beta_k \cdot \tau$ iff β_h matches the same input in $\omega_h \cdot \beta_h \cdot \tau$, since $\beta_h \cdot \beta_k \cdot \tau \approx_{\omega'} \beta_k \cdot \beta_h \cdot \tau$ by Fact (a).
We proceed now to prove (*). We distinguish three cases, according to whether:

i) each $\beta_i \cdot \tau$ is ω_i-pointed, for $i = 1, 2$;
ii) no $\beta_i \cdot \tau$ is ω_i-pointed, for $i = 1, 2$;
iii) $\beta_h \cdot \tau$ is ω_h-pointed and $\beta_k \cdot \tau$ is not ω_k-pointed, for $h, k = 1, 2$ and $h \neq k$.

Case i). Suppose each $\beta_i \cdot \tau$ is ω_i-pointed, for $i = 1, 2$. Then $\beta_1 \lceil_{\omega'} (\beta_2 \lceil_{\omega_2} \tau) \approx_{\omega'} \beta_1 \lceil_{\omega'} \beta_2 \cdot \tau$ and $\beta_2 \lceil_{\omega'} (\beta_1 \lceil_{\omega_1} \tau) \approx_{\omega'} \beta_2 \lceil_{\omega'} \beta_1 \cdot \tau$. By Fact (d) both $\beta_1 \cdot \beta_2 \cdot \tau$ and $\beta_2 \cdot \beta_1 \cdot \tau$ are ω'-pointed. Then $\beta_1 \lceil_{\omega'} \beta_2 \cdot \tau \approx_{\omega'} \beta_1 \cdot \beta_2 \cdot \tau$ and $\beta_2 \lceil_{\omega'} \beta_1 \cdot \tau \approx_{\omega'} \beta_2 \cdot \beta_1 \cdot \tau$. By Fact (a) $\beta_1 \cdot \beta_2 \cdot \tau \approx_{\omega'} \beta_2 \cdot \beta_1 \cdot \tau$.
Case ii). Suppose no $\beta_i \cdot \tau$ is ω_i-pointed, for $i = 1, 2$. Then $\beta_1 \lceil_{\omega'} (\beta_2 \lceil_{\omega_2} \tau) \approx_{\omega'} \beta_1 \lceil_{\omega'} \tau$ and $\beta_2 \lceil_{\omega'} (\beta_1 \lceil_{\omega_1} \tau) \approx_{\omega'} \beta_2 \lceil_{\omega'} \tau$. By Fact (b), no $\beta_i \cdot \tau$ can be ω'-pointed, for $i \in \{1, 2\}$. Hence $\beta_1 \lceil_{\omega'} \tau \approx_{\omega'} \tau \approx_{\omega'} \beta_2 \lceil_{\omega'} \tau$.
Case iii). Suppose $\beta_h \cdot \tau$ is ω_h-pointed and $\beta_k \cdot \tau$ is not ω_k-pointed, for $h, k = 1, 2$ and $h \neq k$.

Then $\beta_h \lceil_{\omega'} (\beta_k \lceil_{\omega_k} \tau) \approx_{\omega'} \beta_h \lceil_{\omega'} \tau$ and $\beta_k \lceil_{\omega'} (\beta_h \lceil_{\omega_h} \tau) \approx_{\omega'} \beta_k \lceil_{\omega'} \beta_h \cdot \tau$. By Fact (c) $\beta_h \cdot \tau$ is ω'-pointed. Hence $\beta_h \lceil_{\omega'} \tau \approx_{\omega'} \beta_h \cdot \tau$. By Fact (d) $\beta_k \cdot \beta_h \cdot \tau$ is not ω'-pointed. Therefore $\beta_k \lceil_{\omega'} \beta_h \cdot \tau \approx_{\omega'} \beta_h \cdot \tau$.

(4) Let $\delta = [\omega, \tau]_\sim$. Since both $\beta_2 \bullet \delta$ and $\beta_2 \bullet (\beta_1 \circ \delta)$ are defined, by Lemma 8.14 both $\beta_2 \blacktriangleright \omega$ and $\beta_2 \blacktriangleright (\beta_1 \triangleright \omega)$ must be defined. Then, by Lemma 8.3(2) $\beta_2 \blacktriangleright (\beta_1 \triangleright \omega) \cong \beta_1 \triangleright (\beta_2 \blacktriangleright \omega)$. So we set $\omega' = \beta_1 \triangleright (\beta_2 \blacktriangleright \omega)$. Let $\omega_1 = \beta_1 \triangleright \omega$. By Definition 8.13(1) we get

$$\delta_1 = \beta_1 \circ \delta = \begin{cases} [\omega_1, \beta_1 \cdot \tau]_\sim & \text{if } \beta_1 \cdot \tau \text{ is } \omega_1\text{-pointed} \\ [\omega_1, \tau]_\sim & \text{otherwise} \end{cases}$$

Let $\omega_2 = \beta_2 \blacktriangleright \omega$. By Definition 8.13(2) we get

$$\delta_2 = \beta_2 \bullet \delta = \begin{cases} [\omega_2, \tau']_\sim & \text{if } \tau \approx_\omega \beta_2 \cdot \tau' \\ [\omega_2, \tau]_\sim & \text{if } \mathsf{play}(\beta_2) \cap \mathsf{play}(\tau) = \emptyset \end{cases}$$

The remainder of this proof is split into two cases, according to the shape of δ_2.

Case $\delta_2 = [\omega_2, \tau]_\sim$. Then $\mathsf{play}(\beta_2) \cap \mathsf{play}(\tau) = \emptyset$. By Definition 8.13(1) we get

$$\beta_1 \circ \delta_2 \quad = \quad \begin{cases} [\omega', \beta_1 \cdot \tau]_\sim & \text{if } \beta_1 \cdot \tau \text{ is } \omega'\text{-pointed} \\ [\omega', \tau]_\sim & \text{otherwise} \end{cases}$$

Since $\mathsf{play}(\beta_2) \cap \mathsf{play}(\beta_1 \cdot \tau) = \emptyset$, by Definition 8.13(2) we get

$$\beta_2 \bullet \delta_1 \quad = \quad \begin{cases} [\omega', \beta_1 \cdot \tau]_\sim & \text{if } \beta_1 \cdot \tau \text{ is } \omega_1\text{-pointed} \\ [\omega', \tau]_\sim & \text{otherwise} \end{cases}$$

We have to show that

$$(**) \quad \beta_1 \cdot \tau \text{ is } \omega'\text{-pointed iff } \beta_1 \cdot \tau \text{ is } \omega_1\text{-pointed}$$

If β_1 is an input, it must be required in τ for both ω'-pointedness and ω_1-pointedness, so this case is obvious.

Let $\beta_1 = \mathsf{pq}!\lambda$.

If β_2 is an output, then $\omega' \cong \omega_1 \cdot \beta_2$ by Definition 8.2(2). Since $\beta_2 \neq \mathsf{pq}!\lambda'$ for all λ', an input β_0 in τ matches β_1 in $\omega_1 \cdot \beta_2 \cdot \beta_1 \cdot \tau$ iff it matches β_1 in $\omega_1 \cdot \beta_1 \cdot \tau$.

If β_2 is an input, then $\omega_1 \cong \overline{\beta_2} \cdot \omega'$ by Definition 8.2(2). If $\beta_2 \neq \mathsf{pq}?\lambda'$ for all λ', then an input β_0 in τ matches β_1 in $\omega' \cdot \beta_1 \cdot \tau$ iff it matches β_1 in $\overline{\beta_2} \cdot \omega' \cdot \beta_1 \cdot \tau$. Let $\beta_2 = \mathsf{pq}?\lambda'$ for some λ'. Since $\mathsf{play}(\beta_2) \cap \mathsf{play}(\tau) \neq \emptyset$, there is no β_0 in τ such that β_0 matches β_1 in $\omega' \cdot \beta_1 \cdot \tau$ or in $\overline{\beta_2} \cdot \omega' \cdot \beta_1 \cdot \tau$. This concludes the proof of (**).

Case $\delta_2 = [\omega_2, \tau']_\sim$. Then $\tau \approx_\omega \beta_2 \cdot \tau'$. By Definition 8.13(1) we get

$$\beta_1 \circ \delta_2 \quad = \quad \begin{cases} [\omega', \beta_1 \cdot \tau']_\sim & \text{if } \beta_1 \cdot \tau' \text{ is } \omega'\text{-pointed} \\ [\omega', \tau']_\sim & \text{otherwise} \end{cases}$$

and, since $\delta = [\omega, \beta_2 \cdot \tau']_\sim$, by the same definition we get

$$\beta_1 \circ \delta \quad = \quad \begin{cases} [\omega_1, \beta_1 \cdot \beta_2 \cdot \tau']_\sim & \text{if } \beta_1 \cdot \beta_2 \cdot \tau' \text{ is } \omega_1\text{-pointed} \\ [\omega_1, \beta_2 \cdot \tau']_\sim & \text{otherwise} \end{cases}$$

We first show that $\beta_2 \cdot \beta_1 \cdot \tau' \approx_{\omega_1} \beta_1 \cdot \beta_2 \cdot \tau'$. Since $\beta_1 \circ \delta$ is defined, $\omega_1 \cdot \beta_1 \cdot \beta_2 \cdot \tau'$ is well formed by Lemma 8.14(1). So β_1 cannot be a matching input for β_2. To show that β_2 cannot be a matching input for β_1 observe that, if it were, then $\beta_1 = \overline{\beta_2}$. Since $\beta_2 \bullet (\beta_1 \circ \delta)$ is defined we have that $\omega_1 \equiv \overline{\beta_2} \cdot \omega'$ by Definition 8.2(2). Therefore β_2 cannot be a matching input for β_1 in $\overline{\beta_2} \cdot \omega' \cdot \beta_1 \cdot \beta_2 \cdot \tau'$, since it is the matching input of the first $\overline{\beta_2}$. From this and $\mathsf{play}(\beta_1) \cap \mathsf{play}(\beta_2) = \emptyset$ we get that $\beta_2 \cdot \beta_1 \cdot \tau' \approx_{\omega_1} \beta_1 \cdot \beta_2 \cdot \tau'$. Therefore

$$\beta_1 \circ \delta \quad = \quad \begin{cases} [\omega_1, \beta_2 \cdot \beta_1 \cdot \tau']_\sim & \text{if } \beta_1 \cdot \beta_2 \cdot \tau' \text{ is } \omega_1\text{-pointed} \\ [\omega_1, \beta_2 \cdot \tau']_\sim & \text{otherwise} \end{cases}$$

and by Definition 8.13(2)

$$\beta_2 \bullet \delta_1 \quad = \quad \begin{cases} [\omega', \beta_1 \cdot \tau']_\sim & \text{if } \beta_1 \cdot \beta_2 \cdot \tau' \text{ is } \omega_1\text{-pointed} \\ [\omega', \tau']_\sim & \text{otherwise} \end{cases}$$

We have to show that

$$(***) \;\; \beta_1 \cdot \tau' \text{ is } \omega'\text{-pointed iff } \beta_1 \cdot \beta_2 \cdot \tau' \text{ is } \omega_1\text{-pointed}$$

Note that β_1 is required in τ' iff it is required in $\beta_2 \cdot \tau'$ since $\mathsf{play}(\beta_2) \cap \mathsf{play}(\beta_1) = \emptyset$. Therefore the result is immediate when β_1 is an input.
Let β_1 be an output.
If β_2 is an output, then $\omega' \cong \omega_1 \cdot \beta_2$ by Definition 8.2(2). Suppose that $\beta_1 \cdot \tau'$ is ω'-pointed, where $\tau' = \tau'_0 \cdot \beta_0 \cdot \tau''_0$ and the shown occurrence of β_0 matches β_1 in $\omega_1 \cdot \beta_2 \cdot \beta_1 \cdot \tau'$. Then, since $\beta_2 \cdot \beta_1 \cdot \tau' \approx_{\omega_1} \beta_1 \cdot \beta_2 \cdot \tau'$, we have that β_0 matches β_1 in $\omega_1 \cdot \beta_1 \cdot \beta_2 \cdot \tau'$. In a similar way we can prove that, if an input β_0 matches β_1 in $\omega_1 \cdot \beta_1 \cdot \beta_2 \cdot \tau'$, then β_0 matches β_1 in $\omega_1 \cdot \beta_2 \cdot \beta_1 \cdot \tau'$.
If β_2 is an input, then $\omega_1 \cong \overline{\beta_2} \cdot \omega'$ by Definition 8.2(2). Suppose that $\beta_1 \cdot \tau'$ is ω'-pointed where $\tau' = \tau'_0 \cdot \beta_0 \cdot \tau''_0$ and the shown occurrence of β_0 matches β_1 in $\omega' \cdot \beta_1 \cdot \tau'$. Then β_0 matches β_1 in $\overline{\beta_2} \cdot \omega' \cdot \beta_1 \cdot \beta_2 \cdot \tau'$, since β_2 is the first input in the trace and it matches the shown occurrence of $\overline{\beta_2}$. In a similar way we can prove that, if an input β_0 matches β_1 in $\overline{\beta_2} \cdot \omega' \cdot \beta_1 \cdot \beta_2 \cdot \tau'$, then β_0 matches β_1 in $\omega' \cdot \beta_1 \cdot \tau'$. Therefore $(***)$ holds. □

The next lemma shows that the retrieval and residual operators on g-events preserve causality and that the retrieval operator preserves conflict. It is the analogous of Lemma 8.6, but without the statement corresponding to Lemma 8.6(4), which is true but not required for later results. The difference is due to the fact that ESs of networks are FESs, while those of simple agts are PESs. This appears clearly when looking at the proof of Theorem 8.12 which uses Lemma 8.6(4), while that of Theorem 8.27 does not need the corresponding property.

Lemma 8.18 . (1) *If $\delta_1 < \delta_2$ and $\beta \circ \delta_1$ is defined, then $\beta \circ \delta_1 < \beta \circ \delta_2$.*
(2) *If $\delta_1 < \delta_2$ and both $\beta \bullet \delta_1$, $\beta \bullet \delta_2$ are defined, then $\beta \bullet \delta_1 < \beta \bullet \delta_2$.*
(3) *If $\delta_1 \# \delta_2$ and both $\beta \circ \delta_1$, $\beta \circ \delta_2$ are defined, then $\beta \circ \delta_1 \# \beta \circ \delta_2$.*

Proof. (1) Since $\delta_1 < \delta_2$ and $\beta \circ \delta_1$ are defined, then also $\beta \circ \delta_2$ is defined. Let $\delta_1 = [\omega, \tau]_\sim$ and $\delta_2 = [\omega, \tau \cdot \tau']_\sim$. If $\beta \circ \delta_1 = [\omega', \tau]_\sim$ and $\beta \circ \delta_2 = [\omega', \tau \cdot \tau']_\sim$ for some ω', then $\beta \circ \delta_1 < \beta \circ \delta_2$.
Let β be an output. Then $\omega \cong \omega' \cdot \beta$. If $\beta \circ \delta_1 = [\omega', \beta \cdot \tau]_\sim$, then it must be $\beta \circ \delta_2 = [\omega', \beta \cdot \tau \cdot \tau']_\sim$. Thus $\beta \circ \delta_1 < \beta \circ \delta_2$. The only other case is $\beta \circ \delta_1 = [\omega', \tau]_\sim$ and $\beta \circ \delta_2 = [\omega', \beta \cdot \tau \cdot \tau']_\sim$. Since $\beta \circ \delta_1 = [\omega', \tau]_\sim$, $\beta \cdot \tau$ is not ω'-pointed, so $\mathsf{play}(\beta) \not\subseteq \mathsf{play}(\tau)$ and τ does not contain the matching input of β. Therefore $\beta \cdot \tau \cdot \tau' \approx_{\omega'} \tau \cdot \beta \cdot \tau'$ and $\beta \circ \delta_2 = [\omega', \beta \cdot \tau \cdot \tau']_\sim = [\omega', \tau \cdot \beta \cdot \tau']_\sim$, so $\beta \circ \delta_1 < \beta \circ \delta_2$.
Let β be an input. If $\beta \circ \delta_1 = [\overline{\beta} \cdot \omega, \beta \cdot \tau]_\sim$, then it must be $\beta \circ \delta_2 = [\overline{\beta} \cdot \omega, \beta \cdot \tau \cdot \tau']_\sim$. We get $\beta \circ \delta_1 < \beta \circ \delta_2$. The only other case is $\beta \circ \delta_1 = [\overline{\beta} \cdot \omega, \tau]_\sim$ and $\beta \circ \delta_2 = [\overline{\beta} \cdot \omega, \beta \cdot \tau \cdot \tau']_\sim$. If $\beta \circ \delta_1 = [\overline{\beta} \cdot \omega, \tau]_\sim$, then $\mathsf{play}(\beta) \not\subseteq \mathsf{play}(\tau)$. Therefore $\beta \cdot \tau \cdot \tau' \approx_{\overline{\beta} \cdot \omega} \tau \cdot \beta \cdot \tau'$ and $\beta \circ \delta_2 = [\overline{\beta} \cdot \omega, \beta \cdot \tau \cdot \tau']_\sim = [\overline{\beta} \cdot \omega, \tau \cdot \beta \cdot \tau']_\sim$, so $\beta \circ \delta_1 < \beta \circ \delta_2$.

(2) Let $\delta_1 = [\omega, \tau]_\sim$ and $\delta_2 = [\omega, \tau \cdot \tau']_\sim$. If $\beta \bullet \delta_1 = [\omega', \tau]_\sim$ and $\beta \bullet \delta_2 = [\omega', \tau \cdot \tau']_\sim$ for some ω', then $\beta \bullet \delta_1 < \beta \bullet \delta_2$.
Let β be an output. If $\tau \approx_\omega \beta \cdot \tau_1$ with $\tau_1 \neq \epsilon$, then $\beta \bullet \delta_1 = [\omega \cdot \beta, \tau_1]_\sim$ and $\beta \bullet \delta_2 = [\omega \cdot \beta, \tau_1 \cdot \tau']_\sim$. Therefore $\beta \bullet \delta_1 < \beta \bullet \delta_2$. Let $\mathsf{play}(\beta) \not\subseteq \mathsf{play}(\tau)$ and $\tau \cdot \tau' \approx_\omega \beta \cdot \tau_2$ with $\tau_2 \neq \epsilon$. This implies $\beta \cdot \tau_2 \approx_\omega \beta \cdot \tau \cdot \tau'_2$ for some τ'_2. It follows that $\tau_2 \approx_{\omega \cdot \beta} \tau \cdot \tau'_2$. Then we get $\beta \bullet \delta_1 = [\omega \cdot \beta, \tau]_\sim$

and $\beta \bullet \delta_2 = [\omega \cdot \beta, \tau_2]_\sim = [\omega \cdot \beta, \tau \cdot \tau_2']_\sim$, which imply $\beta \bullet \delta_1 < \beta \bullet \delta_2$.

Let β be an input. The proof is similar.

(3) Let $\delta_1 = [\omega, \tau]_\sim$ and $\delta_2 = [\omega, \tau']_\sim$ and $\tau \,@\, \mathsf{p} \,\#\, \tau' \,@\, \mathsf{p}$. We select some interesting cases.
Note first that $\tau \,@\, \mathsf{p} \,\#\, \tau' \,@\, \mathsf{p}$ implies $\mathsf{p} \in \mathsf{play}(\tau) \cap \mathsf{play}(\tau')$.
If β is an output , then $\omega \cong \omega' \cdot \beta$. If both $\beta \cdot \tau$ and $\beta \cdot \tau'$ are ω'-pointed or not ω'-pointed, then the result is immediate. If $\beta \cdot \tau$ is ω'-pointed while $\beta \cdot \tau'$ is not ω'-pointed, then $\mathsf{play}(\beta) \not\subseteq \mathsf{play}(\tau')$. This implies $\mathsf{p} \notin \mathsf{play}(\beta)$. Similarly, if β is an input and $\mathsf{play}(\beta) \subseteq \mathsf{play}(\tau)$ while $\mathsf{play}(\beta) \not\subseteq \mathsf{play}(\tau')$, then $\mathsf{p} \notin \mathsf{play}(\beta)$. In both cases we get $(\beta \cdot \tau) \,@\, \mathsf{p} = \tau \,@\, \mathsf{p} \,\#\, \tau' \,@\, \mathsf{p}$, so we conclude $\beta \circ \delta_1 \,\#\, \beta \circ \delta_2$. □

The next lemma shows that the operator $\circ$ builds g-events of a simple agt $\mathsf{G} \parallel \mathcal{M}$ from g-events of the immediate subtypes of G composed in parallel with the appropriate queues, as given by the input/output matching of Figure 3. Symmetrically, $\bullet$ builds g-events of agts whose sgts are subtypes of G starting from g-events of $\mathsf{G} \parallel \mathcal{M}$.

Lemma 8.19 . (1) *If* $\delta \in \mathcal{GE}(\mathsf{G} \parallel \mathcal{M} \cdot \langle \mathsf{p}, \lambda, \mathsf{q} \rangle)$, *then*
$$\mathsf{pq}!\lambda \circ \delta \in \mathcal{GE}(\mathsf{pq}! \boxplus_{i \in I} \lambda_i; \mathsf{G}_i \parallel \mathcal{M}) \text{ where } \lambda = \lambda_k \text{ and } \mathsf{G} = \mathsf{G}_k \text{ for some } k \in I.$$
(2) *If* $\delta \in \mathcal{GE}(\mathsf{G} \parallel \mathcal{M})$, *then* $\mathsf{pq}?\lambda \circ \delta \in \mathcal{GE}(\mathsf{pq}?\lambda; \mathsf{G} \parallel \langle \mathsf{p}, \lambda, \mathsf{q} \rangle \cdot \mathcal{M})$.
(3) *If* $\delta \in \mathcal{GE}(\mathsf{pq}! \boxplus_{i \in I} \lambda_i; \mathsf{G}_i \parallel \mathcal{M})$ *and* $\mathsf{pq}!\lambda_k \bullet \delta$ *is defined, then*
$$\mathsf{pq}!\lambda_k \bullet \delta \in \mathcal{GE}(\mathsf{G}_k \parallel \mathcal{M} \cdot \langle \mathsf{p}, \lambda_k, \mathsf{q} \rangle) \text{ where } k \in I.$$
(4) *If* $\delta \in \mathcal{GE}(\mathsf{pq}?\lambda; \mathsf{G} \parallel \langle \mathsf{p}, \lambda, \mathsf{q} \rangle \cdot \mathcal{M})$ *and* $\mathsf{pq}?\lambda \bullet \delta$ *is defined, then* $\mathsf{pq}?\lambda \bullet \delta \in \mathcal{GE}(\mathsf{G} \parallel \mathcal{M})$.

Proof. (1) By Definition 7.18(1), if $\delta \in \mathcal{GE}(\mathsf{G} \parallel \mathcal{M} \cdot \langle \mathsf{p}, \lambda, \mathsf{q} \rangle)$, then $\delta = \mathsf{ev}(\omega \cdot \mathsf{pq}!\lambda, \tau)$ where $\omega = \mathsf{otr}(\mathcal{M})$ and $\tau \in \mathsf{Tr}^+(\mathsf{G})$. By Lemma 8.16(1) $\mathsf{pq}!\lambda \circ \delta = \mathsf{ev}(\omega, \mathsf{pq}!\lambda \cdot \tau)$. Then, again by Definition 7.18(1), $\mathsf{pq}!\lambda \circ \delta \in \mathcal{GE}(\mathsf{pq}! \boxplus_{i \in I} \lambda_i; \mathsf{G}_i \parallel \mathcal{M})$ where $\lambda = \lambda_k$ and $\mathsf{G} = \mathsf{G}_k$ for some $k \in I$, since $\mathsf{pq}!\lambda_k \cdot \tau \in \mathsf{Tr}^+(\mathsf{pq}! \boxplus_{i \in I} \lambda_i; \mathsf{G}_i)$.

(2) Similar to the proof of (1).

(3) By Definition 7.18(1), if $\delta \in \mathcal{GE}(\mathsf{pq}! \boxplus_{i \in I} \lambda_i; \mathsf{G}_i \parallel \mathcal{M})$, then $\delta = \mathsf{ev}(\omega, \tau)$ where $\omega = \mathsf{otr}(\mathcal{M})$ and $\tau \in \mathsf{Tr}^+(\mathsf{pq}! \boxplus_{i \in I} \lambda_i; \mathsf{G}_i)$, which implies $\tau \approx_\omega \mathsf{pq}!\lambda_i \cdot \tau_i$ with $\tau_i \in \mathsf{Tr}^+(\mathsf{G}_i)$ for some $i \in I$. By hypothesis $\mathsf{pq}!\lambda_k \bullet \delta$ is defined, which implies $\tau \approx_\omega \mathsf{pq}!\lambda_k \cdot \tau_k$ and $\tau_k \neq \epsilon$. Then Lemma 8.16(2) gives $\mathsf{pq}!\lambda_k \bullet \delta = \mathsf{ev}(\omega \cdot \mathsf{pq}!\lambda_k, \tau_k)$. We conclude that $\mathsf{pq}!\lambda_k \bullet \delta \in \mathcal{GE}(\mathsf{G}_k \parallel \mathcal{M} \cdot \langle \mathsf{p}, \lambda_k, \mathsf{q} \rangle)$.

(4) Similar to the proof of (3). □

Like $\lozenge$ and $\blacklozenge$, the operators $\circ$ and $\bullet$ modify g-events in the same way as the transitions in the LTS would do. This is formalised and proved in the following lemma, which is similar to Lemma 8.7.

Lemma 8.20 . *Let* $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\beta} \mathsf{G}' \parallel \mathcal{M}'$. *Then* $\mathsf{otr}(\mathcal{M}) \cong \beta \triangleright \mathsf{otr}(\mathcal{M}')$ *and*
(1) *if* $\delta \in \mathcal{GE}(\mathsf{G}' \parallel \mathcal{M}')$, *then* $\beta \circ \delta \in \mathcal{GE}(\mathsf{G} \parallel \mathcal{M})$;
(2) *if* $\delta \in \mathcal{GE}(\mathsf{G} \parallel \mathcal{M})$ *and* $\beta \bullet \delta$ *is defined, then* $\beta \bullet \delta \in \mathcal{GE}(\mathsf{G}' \parallel \mathcal{M}')$.

Proof. We first show $\mathsf{otr}(\mathcal{M}) \cong \beta \triangleright \mathsf{otr}(\mathcal{M}')$. Let $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\beta} \mathsf{G}' \parallel \mathcal{M}'$. If $\beta = \mathsf{pq}!\lambda$, then by Lemma 6.12(1) $\mathcal{M}' \equiv \mathcal{M} \cdot \langle \mathsf{p}, \lambda, \mathsf{q} \rangle$. Thus $\mathsf{otr}(\mathcal{M}') \cong \mathsf{otr}(\mathcal{M}) \cdot \beta$. If $\beta = \mathsf{pq}?\lambda$, then by Lemma 6.12(2) $\mathcal{M} \equiv \langle \mathsf{p}, \lambda, \mathsf{q} \rangle \cdot \mathcal{M}'$. Thus $\mathsf{otr}(\mathcal{M}) \cong \overline{\beta} \cdot \mathsf{otr}(\mathcal{M}')$. Therefore $\mathsf{otr}(\mathcal{M}) \cong \beta \triangleright \mathsf{otr}(\mathcal{M}')$.

(1) By induction on the inference of the transition $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\beta} \mathsf{G}' \parallel \mathcal{M}'$, see Figure 5.
Base Cases. If the applied rule is [Ext-Out], then $\mathsf{G} = \mathsf{pq}! \boxplus_{i \in I} \lambda_i; \mathsf{G}_i$ and $\beta = \mathsf{pq}!\lambda_k$ and $\mathsf{G}' = \mathsf{G}_k$ and $\mathcal{M}' \equiv \mathcal{M} \cdot \langle \mathsf{p}, \lambda_k, \mathsf{q} \rangle$ for some $k \in I$. By Lemma 8.19(1) $\beta \circ \delta \in \mathcal{GE}(\mathsf{G} \parallel \mathcal{M})$.

If the applied rule is [Ext-In], then $\mathsf{G} = \mathsf{pq}?\lambda;\mathsf{G}'$ and $\beta = \mathsf{pq}?\lambda$ and $\mathcal{M} \equiv \langle \mathsf{p}, \lambda, \mathsf{q}\rangle \cdot \mathcal{M}'$. By Lemma 8.19(2) $\beta \circ \delta \in \mathcal{GE}(\mathsf{G} \parallel \mathcal{M})$.
Inductive Cases. If the last applied rule is [IComm-Out], then $\mathsf{G} = \mathsf{pq}!\boxplus_{i\in I}\lambda_i;\mathsf{G}_i$ and $\mathsf{G}' = \mathsf{pq}!\boxplus_{i\in I}\lambda_i;\mathsf{G}'_i$ and $\mathsf{G}_i \parallel \mathcal{M}\cdot\langle \mathsf{p},\lambda_i,\mathsf{q}\rangle \xrightarrow{\beta} \mathsf{G}'_i \parallel \mathcal{M}'\cdot\langle \mathsf{p},\lambda_i,\mathsf{q}\rangle$ for all $i \in I$ and $\mathsf{p} \notin \mathsf{play}(\beta)$.
By Definition 7.18(1) $\delta \in \mathcal{GE}(\mathsf{G}' \parallel \mathcal{M}')$ implies $\delta = \mathsf{ev}(\omega,\tau)$ where $\omega = \mathsf{otr}(\mathcal{M}')$ and $\tau \in \mathsf{Tr}^+(\mathsf{G}')$. Then $\tau = \mathsf{pq}!\lambda_k\cdot\tau'$ and $\delta = [\omega,\tau_0]_\sim$ with $\tau_0 = (\mathsf{pq}!\lambda_k\cdot\tau')\lceil_\omega \epsilon$ for some $k \in I$ by Definition 7.14. We get either $\tau_0 \approx_\omega \mathsf{pq}!\lambda_k\cdot\tau'_0$ or $\mathsf{p} \notin \mathsf{play}(\tau_0)$ by Definition 7.13. Then $\mathsf{pq}!\lambda_k \bullet \delta$ is defined unless $\tau'_0 = \epsilon$ by Definition 8.13(2). We consider the two cases.
Case $\tau'_0 = \epsilon$. We get $\beta\circ\delta = [\beta \triangleright \omega, \mathsf{pq}!\lambda_k]_\sim$ since $\mathsf{p} \notin \mathsf{play}(\beta)$, which implies $\beta\circ\delta \in \mathcal{GE}(\mathsf{G} \parallel \mathcal{M})$ by Definition 7.18(1).
Case $\tau'_0 \neq \epsilon$ *or* $\mathsf{p} \notin \mathsf{play}(\tau_0)$. Let $\delta' = \mathsf{pq}!\lambda_k \bullet \delta$. By Lemma 8.19(3) $\delta' \in \mathcal{GE}(\mathsf{G}'_k \parallel \mathcal{M}'\cdot\langle \mathsf{p},\lambda_k,\mathsf{q}\rangle)$. By induction $\beta\circ\delta' \in \mathcal{GE}(\mathsf{G}_k \parallel \mathcal{M}\cdot\langle \mathsf{p},\lambda_k,\mathsf{q}\rangle)$. Since δ' is defined, Lemma 8.17(2) implies $\mathsf{pq}!\lambda_k \circ \delta' = \delta$. Since $\beta\circ\delta'$ and $\mathsf{pq}!\lambda_k\circ\delta'$ are defined, by Lemma 8.17(3) and $\mathsf{p}\notin\mathsf{play}(\beta)$ we get $\mathsf{pq}!\lambda_k \circ (\beta\circ\delta') = \beta\circ(\mathsf{pq}!\lambda_k\circ\delta') = \beta\circ\delta$. By Lemma 8.19(1) $\mathsf{pq}!\lambda_k\circ(\beta\circ\delta') \in \mathcal{GE}(\mathsf{G}\parallel\mathcal{M})$. We conclude that $\beta\circ\delta \in \mathcal{GE}(\mathsf{G}\parallel\mathcal{M})$.
If the last applied rule is [IComm-In] the proof is similar.

(2) By induction on the inference of the transition $\mathsf{G}\parallel\mathcal{M} \xrightarrow{\beta} \mathsf{G}'\parallel\mathcal{M}'$, see Figure 5.
Base Cases. If the applied rule is [Ext-Out], then $\mathsf{G} = \mathsf{pq}!\boxplus_{i\in I}\lambda_i;\mathsf{G}_i$ and $\beta = \mathsf{pq}!\lambda_k$ and $\mathsf{G}' = \mathsf{G}_k$ and $\mathcal{M}' \equiv \mathcal{M}\cdot\langle \mathsf{p},\lambda_k,\mathsf{q}\rangle$ for some $k \in I$. By assumption $\beta\bullet\delta$ is defined. By Lemma 8.19(3) $\beta\bullet\delta \in \mathcal{GE}(\mathsf{G}'\parallel\mathcal{M}')$.
If the applied rule is [Ext-In], then $\mathsf{G} = \mathsf{pq}?\lambda;\mathsf{G}'$ and $\beta = \mathsf{pq}?\lambda$ and $\mathcal{M} \equiv \langle \mathsf{p},\lambda,\mathsf{q}\rangle\cdot\mathcal{M}'$. By assumption $\beta\bullet\delta$ is defined. By Lemma 8.19(4) $\beta\bullet\delta \in \mathcal{GE}(\mathsf{G}'\parallel\mathcal{M}')$.
Inductive Cases. If the last applied rule is [IComm-Out], then $\mathsf{G} = \mathsf{pq}!\boxplus_{i\in I}\lambda_i;\mathsf{G}_i$ and $\mathsf{G}' = \mathsf{pq}!\boxplus_{i\in I}\lambda_i;\mathsf{G}'_i$ and $\mathsf{G}_i \parallel \mathcal{M}\cdot\langle \mathsf{p},\lambda_i,\mathsf{q}\rangle \xrightarrow{\beta} \mathsf{G}'_i \parallel \mathcal{M}'\cdot\langle \mathsf{p},\lambda_i,\mathsf{q}\rangle$ for all $i \in I$ and $\mathsf{p} \notin \mathsf{play}(\beta)$.
By Definition 7.18(1) $\delta \in \mathcal{GE}(\mathsf{G}\parallel\mathcal{M})$ implies $\delta = \mathsf{ev}(\omega,\tau)$ where $\omega = \mathsf{otr}(\mathcal{M})$ and $\tau \in \mathsf{Tr}^+(\mathsf{G})$. Then $\tau = \mathsf{pq}!\lambda_k\cdot\tau'$ and $\delta = [\omega,\tau_0]_\sim$ with $\tau_0 = (\mathsf{pq}!\lambda_k\cdot\tau')\lceil_\omega\epsilon$ for some $k \in I$ by Definition 7.14. We get either $\tau_0 \approx_\omega \mathsf{pq}!\lambda_k\cdot\tau'_0$ or $\mathsf{p}\notin\mathsf{play}(\tau_0)$ by Definition 7.13. Then $\mathsf{pq}!\lambda_k\bullet\delta$ is defined unless $\tau'_0 = \epsilon$ by Definition 8.13(2). We consider the two cases.
Case $\tau'_0 = \epsilon$. We get $\beta\bullet\delta = [\beta \blacktriangleright \omega, \mathsf{pq}!\lambda_k]_\sim$ since $\mathsf{play}(\beta)\cap\mathsf{play}(\mathsf{pq}!\lambda_k) = \emptyset$, which implies $\beta\bullet\delta \in \mathcal{GE}(\mathsf{G}'\parallel\mathcal{M}')$ by Definition 7.18(1).
Case $\tau'_0 \neq \epsilon$ *or* $\mathsf{p}\notin\mathsf{play}(\tau_0)$. Let $\delta' = \mathsf{pq}!\lambda_k\bullet\delta$. By Lemma 8.19(3) $\delta' \in \mathcal{GE}(\mathsf{G}_k\parallel\mathcal{M}\cdot\langle \mathsf{p},\lambda_k,\mathsf{q}\rangle)$. By assumption $\beta\bullet\delta$ is defined. We first show that $\beta\bullet\delta'$ is defined. Since $\beta\bullet\delta$ and $\mathsf{pq}!\lambda_k\bullet\delta$ are defined, by Definition 8.13(2) we have four cases:

(a) $\tau_0 \approx_\omega \beta\cdot\tau_1$ for some τ_1 and $\tau_0 \approx_\omega \mathsf{pq}!\lambda_k\cdot\tau'_0$;
(b) $\tau_0 \approx_\omega \beta\cdot\tau_1$ and $\mathsf{p}\notin\mathsf{play}(\tau_0)$;
(c) $\mathsf{play}(\beta)\cap\mathsf{play}(\tau_0) = \emptyset$ and $\tau_0 \approx_\omega \mathsf{pq}!\lambda_k\cdot\tau'_0$;
(d) $\mathsf{play}(\beta)\cap\mathsf{play}(\tau_0) = \emptyset$ and $\mathsf{p}\notin\mathsf{play}(\tau_0)$.

Let $\omega' = \mathsf{pq}!\lambda_k \blacktriangleright \omega = \omega\cdot\mathsf{pq}!\lambda_k$ and $\omega'' = \beta\blacktriangleright\omega'$.
In Case (a) we have $\tau_0 \approx_\omega \beta\cdot\mathsf{pq}!\lambda_k\cdot\tau'_1 \approx_\omega \mathsf{pq}!\lambda_k\cdot\beta\cdot\tau'_1$ for some τ'_1. Let $\tau_2 = \beta\cdot\tau'_1$. Then $\delta = [\omega,\mathsf{pq}!\lambda_k\cdot\tau_2]_\sim$ and therefore $\delta' = [\omega',\tau_2]_\sim = [\omega',\beta\cdot\tau'_1]_\sim$. Hence $\beta\bullet\delta' = [\omega'',\tau'_1]_\sim$.
In Case (b) we have $\delta = [\omega,\beta\cdot\tau_1]_\sim$ and $\mathsf{p}\notin\mathsf{play}(\beta\cdot\tau_1)$. Therefore $\delta' = [\omega',\beta\cdot\tau_1]_\sim$. Hence $\beta\bullet\delta' = [\omega'',\tau_1]_\sim$.
In Case (c) we have $\delta' = [\omega',\tau'_0]_\sim$ and $\beta\bullet\delta' = [\omega'',\tau'_0]_\sim$ since $\mathsf{play}(\beta)\cap\mathsf{play}(\tau_0) = \emptyset$ implies $\mathsf{play}(\beta)\cap\mathsf{play}(\tau'_0) = \emptyset$.
In Case (d) we have $\delta' = [\omega',\tau_0]_\sim$ and $\beta\bullet\delta' = [\omega'',\tau_0]_\sim$.

So in all cases we conclude that $\beta \bullet \delta'$ is defined.
By induction $\beta \bullet \delta' \in \mathcal{GE}(\mathsf{G}'_k \parallel \mathcal{M}' \cdot \langle \mathsf{p}, \lambda_k, \mathsf{q} \rangle)$. By Lemma 8.19(1) $\mathsf{pq}!\lambda_k \circ (\beta \bullet \delta') \in \mathcal{GE}(\mathsf{G}' \parallel \mathcal{M}')$. Since δ' is defined, Lemma 8.17(2) implies $\mathsf{pq}!\lambda_k \circ \delta' = \delta$. Since $\beta \bullet \delta'$ and $\beta \bullet (\mathsf{pq}!\lambda_k \circ \delta')$ are defined and $\mathsf{p} \notin \mathsf{play}(\beta)$, by Lemma 8.17(4) we get $\mathsf{pq}!\lambda_k \circ (\beta \bullet \delta') = \beta \bullet (\mathsf{pq}!\lambda_k \circ \delta') = \beta \bullet \delta$. We conclude that $\beta \bullet \delta \in \mathcal{GE}(\mathsf{G}' \parallel \mathcal{M}')$.
If the last applied rule is [IComm-In] the proof is similar. □

The function gec, which builds a sequence of g-events corresponding to a pair (ω, τ), is simply defined applying the function ev to ω and to prefixes of τ.

Definition 8.21 (Global events from pairs of o-traces and traces)**.** *Let $\tau \neq \epsilon$ be ω-well formed. We define the* sequence of global events corresponding to ω and τ *by*

$$\mathsf{gec}(\omega, \tau) = \delta_1; \cdots; \delta_n$$

where $\delta_i = \mathsf{ev}(\omega, \tau[1 \ldots i])$ for all i, $1 \leq i \leq n$.

The following lemma establishes the soundness of the above definition.

Lemma 8.22 . *If $\tau \neq \epsilon$ is ω-well formed, then:*
(1) *$\tau[1 \ldots i]$ is ω-well formed for all i, $1 \leq i \leq n$;*
(2) *$\mathsf{ev}(\omega, \tau[1 \ldots i])$ is defined and $\mathsf{i/o}(\mathsf{ev}(\omega, \tau[1 \ldots i])) = \tau[i]$ for all i, $1 \leq i \leq n$.*

Proof. The proof of (1) is immediate since by Definitions 7.1 and 7.2 every prefix of an ω-well formed trace is ω-well formed. Fact (2) follows from Fact (1), Definition 7.14 and Lemma 7.15. □

As for the function nec (Lemma 8.9), the g-events in a sequence generated by gec are not in conflict, and we can retrieve τ from $\mathsf{gec}(\omega, \tau)$ by using the function $\mathsf{i/o}$ given in Definition 7.12(2).

Lemma 8.23 . *Let $\tau \neq \epsilon$ be ω-well formed and $\mathsf{gec}(\omega, \tau) = \delta_1; \cdots; \delta_n$.*
(1) *If $1 \leq k, l \leq n$, then $\neg(\delta_k \# \delta_l)$;*
(2) *$\tau[i] = \mathsf{i/o}(\delta_i)$ for all i, $1 \leq i \leq n$.*

Proof. (1) Let $\delta_i = [\omega, \tau_i]_\sim$ for all i, $1 \leq i \leq n$. By Definitions 8.21 and 7.14 $\tau_i = \tau[1 \ldots i] \lceil_\omega \epsilon$. By Definition 7.13 if $\tau_i @ \mathsf{p} \neq \epsilon$, then there are $k_i \leq i$ and τ'_i such that $\mathsf{play}(\tau[k_i]) = \{\mathsf{p}\}$, $\mathsf{p} \notin \mathsf{play}(\tau'_i)$ and $\tau_i = \tau[1 \ldots k_i - 1] \lceil_\omega \tau[k_i] \cdot \tau'_i$. By the same definition all $\tau[j]$ with $j \leq k_i$ and $\mathsf{play}(\tau[j]) = \{\mathsf{p}\}$ occur in τ_i, in the same order as in τ. Theferore $\tau_i @ \mathsf{p}$ is a prefix of $\tau @ \mathsf{p}$ for all p and all i, $1 \leq i \leq n$. This implies that $\tau_h @ \mathsf{p}$ cannot be in conflict with $\tau_l @ \mathsf{p}$ for any p and any h, l, $1 \leq h, l \leq n$.
(2) Immediate from Definitions 8.21, 7.14 and Lemma 7.15. □

The following lemma, together with Lemma 8.22, ensures that $\mathsf{gec}(\omega, \tau)$ is defined when $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\tau} \mathsf{G}' \parallel \mathcal{M}'$ and $\omega = \mathsf{otr}(\mathcal{M})$.

Lemma 8.24 . *If $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\tau} \mathsf{G}' \parallel \mathcal{M}'$ and $\omega = \mathsf{otr}(\mathcal{M})$, then τ is ω-well formed.*

Proof. The proof is by induction on τ.
Case $\tau = \beta$. If $\beta = \mathsf{pq}!\lambda$, then the result is immediate.

If $\beta = \mathsf{pq}?\lambda$, then from $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\beta} \mathsf{G}' \parallel \mathcal{M}'$ we get $\mathcal{M} \equiv \langle \mathsf{p}, \lambda, \mathsf{q} \rangle \cdot \mathcal{M}'$ by Lemma 6.12(2), which implies $\omega \cong \mathsf{pq}!\lambda \cdot \omega'$. Then the trace $\omega \cdot \beta = \mathsf{pq}!\lambda \cdot \omega' \cdot \mathsf{pq}?\lambda$ is

the first input of q from p and $\mathsf{pq}!\lambda$ is the first output of p to q, and therefore $1 \propto^{\omega\cdot\beta} |\omega| + 1$. Hence β is ω-well formed.

Case $\tau = \beta\cdot\tau'$ *with* $\tau' \neq \epsilon$. Let $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\beta} \mathsf{G}'' \parallel \mathcal{M}''$ $\xrightarrow{\tau'}$ $\mathsf{G}' \parallel \mathcal{M}'$ and $\omega' = \mathsf{otr}(\mathcal{M}'')$. By induction τ' is ω'-well formed. If $\beta = \mathsf{pq}!\lambda$, then from $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\beta} \mathsf{G}'' \parallel \mathcal{M}''$ we get $\mathcal{M}'' = \mathcal{M}\cdot\langle \mathsf{p}, \lambda, \mathsf{q}\rangle$ by Lemma 6.12(1). Therefore $\mathsf{otr}(\mathcal{M}'') = \omega\cdot\beta = \omega'$. Since τ' is $(\omega\cdot\beta)$-well formed, i.e. $\omega\cdot\beta\cdot\tau'$ is well formed, we may conclude that $\tau = \beta\cdot\tau'$ is ω-well formed. If $\beta = \mathsf{pq}?\lambda$, as in the base case we get $\mathcal{M} \equiv \langle \mathsf{p}, \lambda, \mathsf{q}\rangle \cdot \mathcal{M}''$ by Lemma 6.12(2), and thus $\omega = \mathsf{pq}!\lambda\cdot\omega'$. We know that τ' is ω'-well formed, i.e. $\omega'\cdot\tau'$ is well formed. Therefore we have that $\mathsf{pq}!\lambda\cdot\omega'\cdot\mathsf{pq}?\lambda\cdot\tau'$ is well formed, since $1 \propto^{\omega\cdot\tau} |\omega| + 1$, and we may conclude that τ is ω-well formed. □

The following lemma mirrors Lemma 8.10.

Lemma 8.25 . (1) *Let* $\tau = \beta\cdot\tau'$ *and* $\omega = \beta \triangleright \omega'$. *If* $\mathsf{gec}(\omega, \tau) = \delta_1; \cdots; \delta_n$ *and* $\mathsf{gec}(\omega', \tau') = \delta'_2; \cdots; \delta'_n$, *then* $\beta \circ \delta'_i = \delta_i$ *for all* i, $2 \leq i \leq n$.
(2) *Let* $\tau = \beta\cdot\tau'$ *and* $\omega' = \beta \blacktriangleright \omega$. *If* $\mathsf{gec}(\omega, \tau) = \delta_1; \cdots; \delta_n$ *and* $\mathsf{gec}(\omega', \tau') = \delta'_2; \cdots; \delta'_n$, *then* $\beta \bullet \delta_i = \delta'_i$ *for all* i, $2 \leq i \leq n$.

Proof. (1) By Definition 8.21 $\delta_i = \mathsf{ev}(\omega, \beta\cdot\tau'[1 \ldots i])$ and $\delta'_i = \mathsf{ev}(\omega', \tau'[1 \ldots i])$ for all i, $2 \leq i \leq n$. Then by Lemma 8.16(1) $\beta \circ \delta'_i = \delta_i$ for all i, $2 \leq i \leq n$.
(2) By Fact (1) and Lemma 8.17(1). □

We end this subsection with the two theorems for simple agts discussed at the beginning of the whole section, which relate the transition sequences of a simple agt with the proving sequences of the associated PES.

Theorem 8.26 . *If* $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\tau} \mathsf{G}' \parallel \mathcal{M}'$, *then* $\mathsf{gec}(\mathsf{otr}(\mathcal{M}), \tau)$ *is a proving sequence in* $\mathcal{S}^{\mathcal{G}}(\mathsf{G} \parallel \mathcal{M})$.

Proof. Let $\omega = \mathsf{otr}(\mathcal{M})$. By Lemma 8.24 τ is ω-well formed. Then by Lemma 8.22 $\mathsf{gec}(\omega, \tau)$ is defined and by Definition 8.21 $\mathsf{gec}(\omega, \tau) = \delta_1; \cdots; \delta_n$, where $\delta_i = \mathsf{ev}(\omega, \tau[1 \ldots i])$ for all i, $1 \leq i \leq n$. We proceed by induction on τ.
Case $\tau = \beta$. In this case, $\mathsf{gec}(\omega, \beta) = \delta_1 = \mathsf{ev}(\omega, \beta)$. By Definition 7.14 we have $\mathsf{ev}(\omega, \beta) = [\omega, \beta\lceil_\omega \epsilon]_\sim$. By Definition 7.13 $[\omega, \beta\lceil_\omega \epsilon]_\sim = [\omega, \beta]_\sim$ since β is ω-well formed.

We use now a further induction on the inference of the transition $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\beta} \mathsf{G}' \parallel \mathcal{M}'$, see Figure 5.
Base Cases. The rule applied is [Ext-Out] or [Ext-In]. Therefore $\beta \in \mathsf{Tr}^+(\mathsf{G})$. By Definition 7.18(1) this implies $\mathsf{ev}(\omega, \beta) \in \mathcal{GE}(\mathsf{G} \parallel \mathcal{M})$.
Inductive Cases. If the last applied Rule is [IComm-Out], then $\mathsf{G} = \mathsf{pq}!\boxplus_{i\in I} \lambda_i; \mathsf{G}_i$ and $\mathsf{G}' = \mathsf{pq}!\boxplus_{i\in I} \lambda_i; \mathsf{G}'_i$ and $\mathsf{G}_i \parallel \mathcal{M}\cdot\langle \mathsf{p}, \lambda_i, \mathsf{q}\rangle \xrightarrow{\beta} \mathsf{G}'_i \parallel \mathcal{M}'\cdot\langle \mathsf{p}, \lambda_i, \mathsf{q}\rangle$ for all $i \in I$ and $\mathsf{p} \notin \mathsf{play}(\beta)$. We have $\mathsf{otr}(\mathcal{M}\cdot\langle \mathsf{p}, \lambda_i, \mathsf{q}\rangle) = \omega\cdot\mathsf{pq}!\lambda_i$. By induction we get $\mathsf{gec}(\omega\cdot\mathsf{pq}!\lambda_i, \beta) = \delta'_i = [\omega\cdot\mathsf{pq}!\lambda_i, \beta]_\sim \in \mathcal{GE}(\mathsf{G}_i \parallel \mathcal{M}\cdot\langle \mathsf{p}, \lambda_i, \mathsf{q}\rangle)$. By Lemma 8.19(1) $\mathsf{pq}!\lambda_i \circ \delta'_i \in \mathcal{GE}(\mathsf{G} \parallel \mathcal{M})$. Now, from $\mathsf{p} \notin \mathsf{play}(\beta)$ it follows that $\mathsf{pq}!\lambda_i$ is not a local cause of β, namely $\neg(\mathsf{req}(1, \mathsf{pq}!\lambda_i\cdot\beta))$. From Lemma 8.24 β is ω-well-formed. So, if β is an input, its matching output must be in ω. Hence $\mathsf{pq}!\lambda_i$ is not a cross-cause of β, namely $\neg(1 + |\omega| \propto^{\omega\cdot\mathsf{pq}!\lambda_i\cdot\beta} 2 + |\omega|)$. Therefore $\mathsf{pq}!\lambda_i\cdot\beta$ is not ω-pointed. By Definition 8.13(1) we get $\mathsf{pq}!\lambda_i \circ \delta'_i = [\omega, \beta]_\sim = \delta_1$. We conclude again that $\delta_1 \in \mathcal{GE}(\mathsf{G} \parallel \mathcal{M})$ and clearly δ_1 is a proving sequence in $\mathcal{S}^{\mathcal{G}}(\mathsf{G} \parallel \mathcal{M})$ since β has no proper prefix.
If the last applied Rule is [IComm-In] the proof is similar.

Case $\tau = \beta \cdot \tau'$ *with* $\tau' \neq \epsilon$. From $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\tau} \mathsf{G}' \parallel \mathcal{M}'$ we get $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\beta} \mathsf{G}'' \parallel \mathcal{M}'' \xrightarrow{\tau'} \mathsf{G}' \parallel \mathcal{M}'$ for some G'', $\mathcal{M}''$. Let $\omega' = \mathsf{otr}(\mathcal{M}'')$. By Lemma 8.24 τ' is ω'-well formed. Thus $\mathsf{gec}(\omega', \tau')$ is defined by Lemma 8.22. Let $\mathsf{gec}(\omega', \tau') = \delta'_2; \cdots; \delta'_n$. By induction $\mathsf{gec}(\omega', \tau')$ is a proving sequence in $\mathcal{S}^{\mathcal{G}}(\mathsf{G}'' \parallel \mathcal{M}'')$. By Lemma 8.25(1) $\delta_j = \beta \circ \delta'_j$ for all $j, 2 \leq j \leq n$. By Lemma 8.20(1) this implies $\delta_j \in \mathcal{GE}(\mathsf{G} \parallel \mathcal{M})$ for all $j, 2 \leq j \leq n$. From the proof of the base case we know that $\delta_1 = [\omega, \beta]_\sim \in \mathcal{GE}(\mathsf{G} \parallel \mathcal{M})$. What is left to show that $\mathsf{gec}(\omega, \tau)$ is a proving sequence in $\mathcal{S}^{\mathcal{G}}(\mathsf{G} \parallel \mathcal{M})$. By Lemma 8.23(1) no two events in this sequence can be in conflict.
Let $\delta \in \mathcal{GE}(\mathsf{G} \parallel \mathcal{M})$ and $\delta < \delta_k$ for some k, $1 \leq k \leq n$. Note that this implies $j > 1$. If $\beta \bullet \delta$ is undefined, then by Definition 8.13(2) either $\delta = \delta_1$ or $\delta = [\omega, \tau]_\sim$ with $\tau \not\asymp_\omega \beta \cdot \tau'$ and $\mathsf{play}(\beta) \subseteq \mathsf{play}(\tau)$. In the first case we are done. In the second case $\tau \,@\, \mathsf{play}(\beta) \,\#\, \beta \,@\, \mathsf{play}(\beta)$, which implies $\delta_1 \,\#\, \delta$. Since $\delta < \delta_k$ and conflict is hereditary, it follows that $\delta_1 \,\#\, \delta_k$, which contradicts what said above. Hence this second case is not possible. If $\beta \bullet \delta$ is defined, by Lemma 8.20(2) $\beta \bullet \delta \in \mathcal{GE}(\mathsf{G}'' \parallel \mathcal{M}'')$ and by Lemma 8.18(2) $\beta \bullet \delta < \beta \bullet \delta_k$. Let $\delta' = \beta \bullet \delta$. By Lemma 8.25(2) $\beta \bullet \delta_j = \delta'_j$ for all $j, 2 \leq j \leq n$. Thus we have $\delta' < \delta'_k$. Since $\mathsf{gec}(\omega', \tau')$ is a proving sequence in $\mathcal{S}^{\mathcal{G}}(\mathsf{G}'' \parallel \mathcal{M}'')$, by Definition 3.6 there is $h < k$ such that $\delta' = \delta'_h$. By Lemma 8.17(2) we derive $\delta = \beta \circ \delta' = \beta \circ \delta'_h = \delta_h$. □

Theorem 8.27 . *If* $\delta_1; \ldots; \delta_n$ *is a proving sequence in* $\mathcal{S}^{\mathcal{G}}(\mathsf{G} \parallel \mathcal{M})$*, then* $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\tau} \mathsf{G}' \parallel \mathcal{M}'$ *where* $\tau = \mathsf{i/o}(\delta_1) \cdot \ldots \cdot \mathsf{i/o}(\delta_n)$.

Proof. The proof is by induction on the length n of the proving sequence. Let $\omega = \mathsf{otr}(\mathcal{M})$.
Case $n = 1$. Let $\mathsf{i/o}(\delta_1) = \beta$. Since δ_1 is the first event of a proving sequence, it can have no causes, so it must be $\delta_1 = [\omega, \beta]_\sim$. We show this case by induction on $d = \mathsf{depth}(\mathsf{G}, \mathsf{play}(\beta))$.
Case $d = 1$. If $\beta = \mathsf{pq}!\lambda$ we have $\mathsf{G} = \mathsf{pq}!\, \boxplus_{i \in I}\, \lambda_i; \mathsf{G}_i$ with $\lambda_k = \lambda$ for some $k \in I$. We deduce $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\beta} \mathsf{G}_k \parallel \mathcal{M} \cdot \langle \mathsf{p}, \lambda, \mathsf{q} \rangle$ by applying Rule [Ext-Out]. If $\beta = \mathsf{pq}?\lambda$ we have $\mathsf{G} = \mathsf{pq}?\lambda; \mathsf{G}'$. Since $\mathsf{G} \parallel \mathcal{M}$ is well formed, by Rule [In] of Figure 3 we get $\mathcal{M} \equiv \langle \mathsf{p}, \lambda, \mathsf{q} \rangle \cdot \mathcal{M}'$. We deduce $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\beta} \mathsf{G}' \parallel \mathcal{M}'$ by applying Rule [Ext-In].
Case $d > 1$. We are in one of the two situations:
(1) $\mathsf{G} = \mathsf{rs}!\, \boxplus_{i \in I}\, \lambda_i; \mathsf{G}_i$ with $\mathsf{r} \notin \mathsf{play}(\beta)$;
(2) $\mathsf{G} = \mathsf{rs}?\lambda'; \mathsf{G}''$ with $\mathsf{s} \notin \mathsf{play}(\beta)$.
In the first case, $\mathsf{r} \notin \mathsf{play}(\beta)$ implies that $\mathsf{rs}!\lambda_i \bullet \delta_1$ is defined for all $i \in I$ by Definition 8.13(2). By Lemma 8.19(3) $\mathsf{rs}!\lambda_i \bullet \delta_1 \in \mathcal{GE}(\mathsf{G}_i \parallel \mathcal{M} \cdot \langle \mathsf{p}, \lambda_i, \mathsf{q} \rangle)$ for all $i \in I$. Lemma 6.4(1) implies $\mathsf{depth}(\mathsf{G}, \mathsf{play}(\beta)) > \mathsf{depth}(\mathsf{G}_i, \mathsf{play}(\beta))$ for all $i \in I$. By induction hypothesis we have $\mathsf{G}_i \parallel \mathcal{M} \cdot \langle \mathsf{p}, \lambda_i, \mathsf{q} \rangle \xrightarrow{\beta} \mathsf{G}'_i \parallel \mathcal{M}' \cdot \langle \mathsf{p}, \lambda_i, \mathsf{q} \rangle$ for all $i \in I$. Then we may apply Rule [IComm-Out] to deduce

$$\mathsf{rs}!\, \boxplus_{i \in I}\, \lambda_i; \mathsf{G}_i \parallel \mathcal{M} \xrightarrow{\beta} \mathsf{rs}!\, \boxplus_{i \in I}\, \lambda_i; \mathsf{G}'_i \parallel \mathcal{M}'$$

In the second case, since $\mathsf{G} \parallel \mathcal{M}$ is well formed we get $\mathcal{M} \equiv \langle \mathsf{r}, \lambda', \mathsf{s} \rangle \cdot \mathcal{M}''$ by Rule [In] of Figure 3. Hence $\omega \cong \mathsf{rs}!\lambda' \cdot \omega'$. This and $\mathsf{s} \notin \mathsf{play}(\beta)$ imply that $\mathsf{rs}?\lambda' \bullet \delta_1$ is defined by Definition 8.13(2). By Lemma 8.19(4) $\mathsf{rs}?\lambda' \bullet \delta_1 \in \mathcal{GE}(\mathsf{G}'' \parallel \mathcal{M}'')$. Lemma 6.4(2) gives $\mathsf{depth}(\mathsf{G}, \mathsf{play}(\beta)) > \mathsf{depth}(\mathsf{G}'', \mathsf{play}(\beta))$. By induction hypothesis $\mathsf{G}'' \parallel \mathcal{M}'' \xrightarrow{\beta} \mathsf{G}''' \parallel \mathcal{M}'''$. Then we may apply Rule [IComm-In] to deduce

$$\mathsf{rs}?\lambda'; \mathsf{G}'' \parallel \langle \mathsf{r}, \lambda', \mathsf{s} \rangle \cdot \mathcal{M}'' \xrightarrow{\beta} \mathsf{rs}?\lambda'; \mathsf{G}''' \parallel \langle \mathsf{r}, \lambda', \mathsf{s} \rangle \cdot \mathcal{M}'''$$

Case $n > 1$. Let $\mathsf{i/o}(\delta_1) = \beta$, and $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\beta} \mathsf{G}'' \parallel \mathcal{M}''$ be the corresponding transition as obtained from the base case. We show that $\beta \bullet \delta_j$ is defined for all $j, 2 \leq j \leq n$. If $\beta \bullet \delta_k$ were

undefined for some k, $2 \leq k \leq n$, then by Definition 8.13(2) either $\delta_k = \delta_1$ or $\delta_k = [\omega, \tau]_\sim$ with $\tau \not\approx_\omega \beta \cdot \tau'$ and $\mathsf{play}(\beta) \subseteq \mathsf{play}(\tau)$. In the second case $\beta @ \mathsf{play}(\beta) \# \tau @ \mathsf{play}(\beta)$, which implies $\delta_k \# \delta_1$. So both cases are impossible. If $\beta \bullet \delta_j$ is defined, by Lemma 8.20(2) we may define $\delta'_j = \beta \bullet \delta_j \in \mathcal{GE}(\mathsf{G}'' \parallel \mathcal{M}'')$ for all j, $2 \leq j \leq n$. We show that $\delta'_2; \cdots ; \delta'_n$ is a proving sequence in $\mathcal{S}^{\mathcal{G}}(\mathsf{G}'' \parallel \mathcal{M}'')$. By Lemma 8.17(2) $\delta_j = \beta \circ \delta'_j$ for all j, $2 \leq j \leq n$. Then by Lemma 8.18(3) no two events in this sequence can be in conflict.
Let $\delta \in \mathcal{GE}(\mathsf{G}'' \parallel \mathcal{M}'')$ and $\delta < \delta'_h$ for some h, $2 \leq h \leq n$. By Lemma 8.20(1) $\beta \circ \delta$ and $\beta \circ \delta'_h$ belong to $\mathcal{GE}(\mathsf{G} \parallel \mathcal{M})$. By Lemma 8.18(1) $\beta \circ \delta < \beta \circ \delta'_h$. By Lemma 8.17(2) $\beta \circ \delta'_h = \delta_h$. Let $\delta' = \beta \circ \delta$. Then $\delta' < \delta_h$ implies, by Definition 3.6 and the fact that $\mathcal{S}^{\mathcal{G}}(\mathsf{G} \parallel \mathcal{M})$ is a PES, that there is $l < h$ such that $\delta' = \delta_l$. By Lemma 8.17(1) we get $\delta = \beta \bullet \delta' = \beta \bullet \delta_l = \delta'_l$.
We have shown that $\delta'_2; \cdots ; \delta'_n$ is a proving sequence in $\mathcal{S}^{\mathcal{G}}(\mathsf{G}'' \parallel \mathcal{M}'')$. By induction $\mathsf{G}'' \parallel \mathcal{M}'' \xrightarrow{\tau'} \mathsf{G}' \parallel \mathcal{M}'$ where $\tau' = \mathsf{i/o}(\delta'_2) \cdot \ldots \cdot \mathsf{i/o}(\delta'_n)$. Let $\tau = \mathsf{i/o}(\delta_1) \cdot \ldots \cdot \mathsf{i/o}(\delta_n)$. Since $\mathsf{i/o}(\delta'_j) = \mathsf{i/o}(\delta_j)$ for all $j, 2 \leq j \leq n$, we have $\tau = \beta \cdot \tau'$. Hence $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\beta} \mathsf{G}'' \parallel \mathcal{M}'' \xrightarrow{\tau'} \mathsf{G}' \parallel \mathcal{M}'$ is the required transition sequence. □

8.3. **Isomorphism.**

We are finally able to show that the ES interpretation of a network is equivalent, when the session is typable, to the ES interpretation of its simple agt.

To prove our main theorem, we will also use the following separation result from [BC91] (Lemma 2.8 p. 12). Recall from Section 3 that $\mathcal{C}(S)$ denotes the set of configurations of S.

Lemma 8.28 (Separation [BC91])**.** *Let $S = (E, \prec, \#)$ be a flow event structure and $X, X' \in \mathcal{C}(S)$ be such that $X \subset X'$. Then there exist $e \in X' \backslash X$ such that $X \cup \{e\} \in \mathcal{C}(S)$.*

We may now establish the isomorphism between the domain of configurations of the FES of a typable network and the domain of configurations of the PES of its simple agt. In the proof of this result, we will use the characterisation of configurations as proving sequences, as given in Proposition 3.7. We will also take the freedom of writing $\rho_1; \cdots ; \rho_n \in \mathcal{C}(\mathcal{S}^{\mathcal{N}}(\mathsf{N} \parallel \mathcal{M}))$ to mean that $\rho_1; \cdots ; \rho_n$ is a proving sequence such that $\{\rho_1, \ldots, \rho_n\} \in \mathcal{C}(\mathcal{S}^{\mathcal{N}}(\mathsf{N} \parallel \mathcal{M}))$, and similarly for $\delta_1; \cdots ; \delta_n \in \mathcal{C}(\mathcal{S}^{\mathcal{G}}(\mathsf{G} \parallel \mathcal{M}))$.

Theorem 8.29 . *If $\vdash \mathsf{N} \parallel \mathcal{M} : \mathsf{G} \parallel \mathcal{M}$, then $\mathcal{D}(\mathcal{S}^{\mathcal{N}}(\mathsf{N} \parallel \mathcal{M})) \simeq \mathcal{D}(\mathcal{S}^{\mathcal{G}}(\mathsf{G} \parallel \mathcal{M}))$.*

Proof. Let $\omega = \mathsf{otr}(\mathcal{M})$. We start by constructing a bijection between the proving sequences of $\mathcal{S}^{\mathcal{N}}(\mathsf{N} \parallel \mathcal{M})$ and the proving sequences of $\mathcal{S}^{\mathcal{G}}(\mathsf{G} \parallel \mathcal{M})$. By Theorem 8.12, if $\rho_1; \cdots ; \rho_n \in \mathcal{C}(\mathcal{S}^{\mathcal{N}}(\mathsf{N} \parallel \mathcal{M}))$, then $\mathsf{N} \parallel \mathcal{M} \xrightarrow{\tau} \mathsf{N}' \parallel \mathcal{M}'$ where $\tau = \mathsf{i/o}(\rho_1) \cdot \cdots \cdot \mathsf{i/o}(\rho_n)$. By applying iteratively Subject Reduction (Theorem 6.18), we obtain

$$\mathsf{G} \parallel \mathcal{M} \xrightarrow{\tau} \mathsf{G}' \parallel \mathcal{M}' \text{ and } \vdash \mathsf{N}' \parallel \mathcal{M}' : \mathsf{G}' \parallel \mathcal{M}'$$

By Theorem 8.26, we get $\mathsf{gec}(\omega, \tau) \in \mathcal{C}(\mathcal{S}^{\mathcal{G}}(\mathsf{G} \parallel \mathcal{M}))$.
By Theorem 8.27, if $\delta_1; \cdots ; \delta_n \in \mathcal{C}(\mathcal{S}^{\mathcal{G}}(\mathsf{G} \parallel \mathcal{M}))$, then $\mathsf{G} \parallel \mathcal{M} \xrightarrow{\tau} \mathsf{G}' \parallel \mathcal{M}'$, where $\tau = \mathsf{i/o}(\delta_1) \cdots \mathsf{i/o}(\delta_n)$. By applying iteratively Session Fidelity (Theorem 6.19), we obtain

$$\mathsf{N} \parallel \mathcal{M} \xrightarrow{\tau} \mathsf{N}' \parallel \mathcal{M}' \text{ and } \vdash \mathsf{N}' \parallel \mathcal{M}' : \mathsf{G}' \parallel \mathcal{M}'$$

By Theorem 8.11, we get $\mathsf{nec}(\omega, \tau) \in \mathcal{C}(\mathcal{S}^{\mathcal{N}}(\mathsf{N} \parallel \mathcal{M}))$.
Therefore we have a bijection between $\mathcal{D}(\mathcal{S}^{\mathcal{N}}(\mathsf{N} \parallel \mathcal{M}))$ and $\mathcal{D}(\mathcal{S}^{\mathcal{G}}(\mathsf{G} \parallel \mathcal{M}))$, given by

$\mathsf{nec}(\omega,\tau) \leftrightarrow \mathsf{gec}(\omega,\tau)$ for any τ generated by the (bisimilar) LTSs of $\mathsf{N} \parallel \mathcal{M}$ and $\mathsf{G} \parallel \mathcal{M}$. We now show that this bijection preserves inclusion of configurations. By Lemma 8.28 it is enough to prove that if $\rho_1;\cdots;\rho_n \in \mathcal{C}(\mathcal{S}^{\mathcal{N}}(\mathsf{N} \parallel \mathcal{M}))$ is mapped to $\delta_1;\cdots;\delta_n \in \mathcal{C}(\mathcal{S}^{\mathcal{G}}(\mathsf{G} \parallel \mathcal{M}))$, then $\rho_1;\cdots;\rho_n;\rho \in \mathcal{C}(\mathcal{S}^{\mathcal{N}}(\mathsf{N} \parallel \mathcal{M}))$ iff $\delta_1;\cdots;\delta_n;\delta \in \mathcal{C}(\mathcal{S}^{\mathcal{G}}(\mathsf{G} \parallel \mathcal{M}))$, where $\delta_1;\cdots;\delta_n;\delta$ is the image of $\rho_1;\cdots;\rho_n;\rho$ under the bijection. So, suppose $\rho_1;\cdots;\rho_n \in \mathcal{C}(\mathcal{S}^{\mathcal{N}}(\mathsf{N} \parallel \mathcal{M}))$ and $\delta_1;\cdots;\delta_n \in \mathcal{C}(\mathcal{S}^{\mathcal{G}}(\mathsf{G} \parallel \mathcal{M}))$ are such that

$$\rho_1;\cdots;\rho_n = \mathsf{nec}(\omega,\tau) \leftrightarrow \mathsf{gec}(\omega,\tau) = \delta_1;\cdots;\delta_n$$

Then $\mathsf{i/o}(\rho_1)\cdots\mathsf{i/o}(\rho_n) = \tau = \mathsf{i/o}(\delta_1)\cdots\mathsf{i/o}(\delta_n)$.

By Theorem 8.12, if $\rho_1;\cdots;\rho_n;\rho \in \mathcal{C}(\mathcal{S}^{\mathcal{N}}(\mathsf{N} \parallel \mathcal{M}))$ with $\mathsf{i/o}(\rho) = \beta$, then

$$\mathsf{N} \parallel \mathcal{M} \xrightarrow{\tau\cdot\beta} \mathsf{N}' \parallel \mathcal{M}'$$

By applying iteratively Subject Reduction (Theorem 6.18) we get

$$\mathsf{G} \parallel \mathcal{M} \xrightarrow{\tau\cdot\beta} \mathsf{G}' \parallel \mathcal{M}' \text{ and } \vdash \mathsf{N}' \parallel \mathcal{M}' : \mathsf{G}' \parallel \mathcal{M}'$$

We conclude that $\mathsf{gec}(\omega,\tau\cdot\beta) \in \mathcal{C}(\mathcal{S}^{\mathcal{G}}(\mathsf{G} \parallel \mathcal{M}))$ by Theorem 8.26.

By Theorem 8.27, if $\delta_1;\cdots;\delta_n;\delta \in \mathcal{C}(\mathcal{S}^{\mathcal{G}}(\mathsf{G} \parallel \mathcal{M}))$ with $\mathsf{i/o}(\delta) = \beta$, then

$$\mathsf{G} \parallel \mathcal{M} \xrightarrow{\tau\cdot\beta} \mathsf{G}' \parallel \mathcal{M}'$$

By applying iteratively Session Fidelity (Theorem 6.19) we get

$$\mathsf{N} \parallel \mathcal{M} \xrightarrow{\tau\cdot\beta} \mathsf{N}' \parallel \mathcal{M}' \text{ and } \vdash \mathsf{N}' \parallel \mathcal{M}' : \mathsf{G}' \parallel \mathcal{M}'$$

We conclude that $\mathsf{nec}(\omega,\tau\cdot\beta) \in \mathcal{C}(\mathcal{S}^{\mathcal{N}}(\mathsf{N} \parallel \mathcal{M}))$ by Theorem 8.11. □

9. Related Work and Conclusions

Session types, as originally proposed in [HVK98, HYC16] for binary sessions, are grounded on types for the π-calculus. Early proposals for typing channels in the π-calculus include simple sorts [Mil91], input/output types [PS96] and usage types [Kob05]. In particular, the notion of progress for multiparty sessions [DY11, CDCYP16] is inspired by the notion of lock-freedom as developed for the π-calculus in [Kob02, Kob06]. The more recent work [DGS12] provides further evidence of the strong relationship between binary session types and channel types in the linear π-calculus. The notion of lock-freedom for the linear π-calculus was also revisited in [Pad15].

Multiparty sessions disciplined by global types were introduced in the keystone papers [HYC08, HYC16]. These papers, as well as most subsequent work on multiparty session types (for a survey see [HLV+16]), were based on more expressive session calculi than the one we use here, where sessions may be interleaved and participants exchange pairs of labels and values. In that more general setting, global types are projected onto session types and in turn session types are assigned to processes. Here, instead, we consider only single sessions and pure label exchange: this allows us to project global types directly to processes, as in [SDC19], where the considered global types are those of [HYC16]. We chose to concentrate on this very simple calculus, as our working plan was already quite challenging. A discussion on possible extensions of our work to more expressive calculi may be found at the end of this section.

Standard global types are too restrictive for typing processes which communicate asynchronously. A powerful typability extension is obtained by the use of the subtyping relation given in [MYH09]. This subtyping allows inputs and outputs to be exchanged,

stating that anticipating outputs is better. The rationale is that outputs are not blocking, while inputs are blocking in asynchronous communication. Unfortunately, this subtyping is undecidable [BCZ17, LY17], and thus type systems equipped with this subtyping are not effective. Decidable restrictions of this subtyping relation have been proposed [BCZ17, LY17, BCZ18]. In particular, subtyping is decidable when both internal and external choices are forbidden in one of the two compared processes [BCZ17]. This result is improved in [BCZ18], where both the subtype and the supertype can contain either internal or external choices. More interestingly, the work [BCL$^+$19] presents a sound (though not complete) algorithm for checking asynchronous subtyping as defined in [CDSY17]. A very elegant formulation of asynchronous subtyping is given in [GPP$^+$21]: it allows the authors to show that any extension of this subtyping would be unsound. In the present paper we achieve a gain in typability for asynchronous networks by using a more fine-grained syntax for global types. Our type system is decidable, since projection is computable and the preorder on processes is decidable. Notice that there are networks that can be typed using the algorithm in [CDSY17] but cannot be typed in our system, like the video streaming example discussed in that paper.

Since their introduction in [Win80] and [NPW81], Event Structures have been widely used to give semantics to process calculi. Several ES interpretations of Milner's calculus CCS have been proposed, using various classes of ESs: Stable ESs in [Win82], Prime ESs or variations of them in [BC87, DDNM88, DDM90], and Flow ESs in [BC88, GG04]. Other calculi such as TCSP (Theoretical CSP [BHR84, Old86]) and LOTOS have been provided respectively with a PES semantics [LG91, BM94] and with a Bundle ES semantics [Lan93, Kat96]. More recently, ES semantics have been investigated also for the π-calculus [CVY07, VY10, CVY12, Cri15, CKV15, CKV16]. We refer the reader to our companion paper [CDG19a] for a more extensive discussion on ES semantics for process calculi.

It is noteworthy that all the above-mentioned ES semantics were given for calculi with synchronous communication. This is perhaps not surprising since ESs are generally equipped with a *synchronisation algebra* when modelling process calculi, and a communication is represented by a single event resulting from the synchronisation of two events. This is also the reason why, in our previous paper [CDG19a], we started by considering an ES semantics for a synchronous session calculus with standard global types. In the present paper, instead, as in [LMT20], a communication is represented by two distinct events in the ES, one for the output and the other for the matching input. Defining the "matching" relation on events, be they events of the FES of a network or events of the PES of an agt, is not entirely trivial.

While agts are interpreted as PESs, the simplest kind of ES, networks are interpreted as FESs, a subclass of Winskel's Stable Event Structures [Win88] that allows for disjunctive causality and therefore provides a more compact representation of networks in the presence of forking computations. Such feature of FESs may be seen at work in Example 5.11, where the event of participant r has two conflicting causes. In a PES this event would have to be replaced by two distinct events, one for each of its two global histories (this is indeed what happens in the PES of the associated agt).

This work builds on the companion paper [CDG19a], where synchronous rather than asynchronous communication was considered. In that paper too, networks were interpreted as FESs, and global types, which were the standard ones, were interpreted as PESs. The key result was again an isomorphism between the configuration domain of the FES of a

typed network and that of the PES of its global type. Thus, the present paper completes the picture set up in [CDG19a] by exploring the "asynchronous side" of the same construction.

As regards future work, we already sketched some possible directions in [CDG19a], including the investigation of reversibility, which would benefit from previous work on reversible session calculi [TY14, TY16, MP17a, MP17b, NY17, CDG19b] and reversible Event Structures [PU15, CKV16, GPY17, GPY18]. We also plan to investigate the extension of our asynchronous calculus with delegation. While delegation is usually defined in session calculi with channels, and modelled using the channel passing mechanism of the π-calculus, an alternative notion of delegation for a session calculus without channels, called "internal delegation", was proposed in [CDGH20]. Hence, one possibility would be to investigate internal delegation in our asynchronous calculus without channels; another option would be to depart from our present calculus, admittedly very simple, and investigate the classical notion of delegation in an asynchronous calculus with channels. Note that delegation remains essentially a synchronous mechanism, even in the asynchronous setting: indeed, unlike ordinary outputs that become non-blocking, delegation remains blocking for the principal, who has to wait until the deputy returns the delegation to be able to proceed. As a matter of fact, this is quite reasonable: not only does it prevent the issue of "power vacancy" that would arise if the role of the principal disappeared from the network for some time, but it also seems natural to assume that the principal delegates a task only when she has the guarantee that the deputy will accept it, and that both of them reside in the same locality (where communication may be assumed to be synchronous).

Acknowledgment. We are indebted to Francesco Dagnino for suggesting a simplification in the definition of input/output matching for asynchronous global types.

References

[ABB+16] Davide Ancona, Viviana Bono, Mario Bravetti, Joana Campos, Giuseppe Castagna, Pierre-Malo Deniélou, Simon J. Gay, Nils Gesbert, Elena Giachino, Raymond Hu, Einar Broch Johnsen, Francisco Martins, Viviana Mascardi, Fabrizio Montesi, Rumyana Neykova, Nicholas Ng, Luca Padovani, Vasco T. Vasconcelos, and Nobuko Yoshida. Behavioral types in programming languages. *Foundations and Trends in Programming Languages*, 3(2-3):95–230, 2016.

[BC87] Gérard Boudol and Ilaria Castellani. On the semantics of concurrency: partial orders and transition systems. In Hartmut Ehrig, Robert A. Kowalski, Giorgio Levi, and Ugo Montanari, editors, *TAPSOFT*, volume 249 of *LNCS*, pages 123–137. Springer, 1987.

[BC88] Gérard Boudol and Ilaria Castellani. Permutation of transitions: an event structure semantics for CCS and SCCS. In Jaco W. de Bakker, Willem P. de Roever, and Grzegorz Rozenberg, editors, *REX: Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, volume 354 of *LNCS*, pages 411–427. Springer, 1988.

[BC91] Gérard Boudol and Ilaria Castellani. Flow models of distributed computations: event structures and nets. Research Report 1482, INRIA, 1991.

[BC94] Gérard Boudol and Ilaria Castellani. Flow models of distributed computations: three equivalent semantics for CCS. *Information and Computation*, 114(2):247–314, 1994.

[BCL+19] Mario Bravetti, Marco Carbone, Julien Lange, Nobuko Yoshida, and Gianluigi Zavattaro. A sound algorithm for asynchronous session subtyping. In Wan J. Fokkink and Rob van Glabbeek, editors, *CONCUR*, volume 140 of *LIPIcs*, pages 38:1–38:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[BCZ17] Mario Bravetti, Marco Carbone, and Gianluigi Zavattaro. Undecidability of asynchronous session subtyping. *Information and Computation*, 256:300–320, 2017.

[BCZ18] Mario Bravetti, Marco Carbone, and Gianluigi Zavattaro. On the boundary between decidability and undecidability of asynchronous session subtyping. *Theoretical Computer Science*, 722:19–51, 2018.

[BHR84] S. Brookes, C.A.R. Hoare, and A. Roscoe. A theory of communicating sequential processes. *Journal of ACM*, 31(3):560–599, 1984.

[BM94] Christel Baier and Mila E. Majster-Cederbaum. The connection between an event structure semantics and an operational semantics for TCSP. *Acta Informatica*, 31(1):81–104, 1994.

[CC13] Felice Cardone and Mario Coppo. Recursive types. In Henk Barendregt, Wil Dekkers, and Richard Statman, editors, *Lambda Calculus with Types*, Perspectives in Logic, pages 377–576. Cambridge University Press, 2013.

[CDCYP16] Mario Coppo, Mariangiola Dezani-Ciancaglini, Nobuko Yoshida, and Luca Padovani. Global progress for dynamically interleaved multiparty sessions. *Mathematical Structures in Computer Science*, 26(2):238–302, 2016.

[CDG19a] Ilaria Castellani, Mariangiola Dezani-Ciancaglini, and Paola Giannini. Event structure semantics for multiparty sessions. In Michele Boreale, Flavio Corradini, Michele Loreti, and 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*, volume 11665 of *LNCS*, pages 340–363. Springer, 2019.

[CDG19b] Ilaria Castellani, Mariangiola Dezani-Ciancaglini, and Paola Giannini. Reversible sessions with flexible choices. *Acta Informatica*, 56(7):553–583, 2019.

[CDGH20] Ilaria Castellani, Mariangiola Dezani-Ciancaglini, Paola Giannini, and Ross Horne. Global Types with Internal Delegation. *Theoretical Computer Science*, 807:128–153, 2020.

[CDSY17] Tzu-Chun Chen, Mariangiola Dezani-Ciancaglini, Alceste Scalas, and Nobuko Yoshida. On the preciseness of subtyping in session types. *Logical Methods in Computer Science*, 13(2), 2017.

[CKV15] Ioana Cristescu, Jean Krivine, and Daniele Varacca. Rigid families for CCS and the π-calculus. In Martin Leucker, Camilo Rueda, and Frank D. Valencia, editors, *ICTAC*, volume 9399 of *LNCS*, pages 223–240. Springer, 2015.

[CKV16] Ioana Cristescu, Jean Krivine, and Daniele Varacca. Rigid families for the reversible π-calculus. In Simon J. Devitt and Ivan Lanese, editors, *Reversible Computation*, volume 9720 of *LNCS*, pages 3–19. Springer, 2016.

[Cou83] Bruno Courcelle. Fundamental properties of infinite trees. *Theoretical Computer Science*, 25:95–169, 1983.

[CP10] Luís Caires and Frank Pfenning. Session types as intuitionistic linear propositions. In Paul Gastin and François Laroussinie, editors, *CONCUR*, volume 6269 of *LNCS*, pages 222–236. Springer, 2010.

[CPT16] Luís Caires, Frank Pfenning, and Bernardo Toninho. Linear logic propositions as session types. *Mathematical Structures in Computer Science*, 26(3):367–423, 2016.

[Cri15] Ioana Cristescu. *Operational and denotational semantics for the reversible π-calculus*. PhD thesis, University Paris Diderot - Paris 7, 2015.

[CVY07] Silvia Crafa, Daniele Varacca, and Nobuko Yoshida. Compositional event structure semantics for the internal π-calculus. In Luís Caires and Vasco T. Vasconcelos, editors, *CONCUR*, volume 4703 of *LNCS*, pages 317–332. Springer, 2007.

[CVY12] Silvia Crafa, Daniele Varacca, and Nobuko Yoshida. Event structure semantics of parallel extrusion in the π-calculus. In Lars Birkedal, editor, *FOSSACS*, volume 7213 of *LNCS*, pages 225–239. Springer, 2012.

[CZ97] Ilaria Castellani and Guo Qiang Zhang. Parallel product of event structures. *Theoretical Computer Science*, 179(1-2):203–215, 1997.

[DCGJ+16] Mariangiola Dezani-Ciancaglini, Silvia Ghilezan, Svetlana Jaksic, Jovanka Pantovic, and Nobuko Yoshida. Precise subtyping for synchronous multiparty sessions. In *PLACES*, volume 203 of *EPTCS*, pages 29 – 44. Open Publishing Association, 2016.

[DDM90] Pierpaolo Degano, Rocco De Nicola, and Ugo Montanari. A partial ordering semantics for CCS. *Theoretical Computer Science*, 75(3):223–262, 1990.

[DDNM88] Pierpaolo Degano, Rocco De Nicola, and Ugo Montanari. On the consistency of truly concurrent operational and denotational semantics. In Ashok K. Chandra, editor, *LICS*, pages 133–141. IEEE Computer Society Press Press, 1988.

[DGS12] Ornela Dardha, Elena Giachino, and Davide Sangiorgi. Session types revisited. In Danny De Schreye, Gerda Janssens, and Andy King, editors, *PPDP*, pages 139–150. ACM, 2012.

[DY11] Pierre-Malo Deniélou and Nobuko Yoshida. Dynamic multirole session types. In Mooly Sagiv Thomas Ball, editor, *POPL*, pages 435–446. ACM Press, 2011.

[DY12] Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty session types meet communicating automata. In Helmut Seidl, editor, *ESOP*, volume 7211 of *LNCS*, pages 194–213. Springer, 2012.

[GG04] Rob J. van Glabbeek and Ursula Goltz. Well-behaved flow event structures for parallel composition and action refinement. *Theoretical Computer Science*, 311(1-3):463–478, 2004.

[GPP+21] Silvia Ghilezan, Jovanka Pantović, Ivan Prokić, Alceste Scalas, and Nobuko Yoshida. Precise subtyping for asynchronous multiparty sessions. In *POPL*. ACM Press, 2021. to appear.

[GPY17] Eva Graversen, Iain Phillips, and Nobuko Yoshida. Towards a categorical representation of reversible event structures. In Vasco T. Vasconcelos and Philipp Haller, editors, *PLACES*, volume 246 of *EPTCS*, pages 49–60. Open Publishing Association, 2017.

[GPY18] Eva Graversen, Iain Phillips, and Nobuko Yoshida. Event structure semantics of (controlled) reversible CCS. In Jarkko Kari and Irek Ulidowski, editors, *Reversible Computation*, volume 11106 of *LNCS*, pages 122–102. Springer, 2018.

[HLV+16] Hans Hüttel, Ivan Lanese, Vasco T. Vasconcelos, Luís Caires, Marco Carbone, Pierre-Malo Deniélou, Dimitris Mostrous, Luca Padovani, António Ravara, Emilio Tuosto, Hugo Torres Vieira, and Gianluigi Zavattaro. Foundations of session types and behavioural contracts. *ACM Computing Surveys*, 49(1):3:1–3:36, 2016.

[HVK98] Kohei Honda, Vasco T. Vasconcelos, and Makoto Kubo. Language primitives and type discipline for structured communication-based programming. In Chris Hankin, editor, *ESOP*, volume 1381 of *LNCS*, pages 122–138. Springer, 1998.

[HYC08] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In George C. Necula and Philip Wadler, editors, *POPL*, pages 273–284. ACM Press, 2008.

[HYC16] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. *Journal of ACM*, 63(1):9:1–9:67, 2016.

[Kat96] Joost-Pieter Katoen. *Quantitative and qualitative extensions of event structures*. PhD thesis, University of Twente, 1996.

[Kob02] Naoki Kobayashi. A type system for lock-free processes. *Information and Computation*, 177(2):122–159, 2002.

[Kob05] Naoki Kobayashi. Type-based information flow analysis for the pi-calculus. *Acta Informatica*, 42(4-5):291–347, 2005.

[Kob06] Naoki Kobayashi. A new type system for deadlock-free processes. In Christel Baier and Holger Hermanns, editors, *CONCUR*, volume 4137 of *LNCS*, pages 233–247. Springer, 2006.

[Lan93] Rom Langerak. Bundle event structures: a non-interleaving semantics for LOTOS. In Michel Diaz and Roland Groz, editors, *Formal Description Techniques for Distributed Systems and Communication Protocols*, pages 331–346. North-Holland, 1993.

[LG91] Rita Loogen and Ursula Goltz. Modelling nondeterministic concurrent processes with event structures. *Fundamenta Informaticae*, 14(1):39–74, 1991.

[LMT20] Ugo de' Liguoro, Hernán C. Melgratti, and Emilio Tuosto. Towards refinable choreographies. In Julien Lange, Anastasia Mavridou, Larisa Safina, and Alceste Scalas, editors, *ICE*, volume 324 of *EPTCS*, pages 61–77. Open Publishing Association, 2020.

[LTY15] Julien Lange, Emilio Tuosto, and Nobuko Yoshida. From communicating machines to graphical choreographies. In Sriram K. Rajamani and David Walker, editors, *POPL*, pages 221–232. ACM Press, 2015.

[LY17] Julien Lange and Nobuko Yoshida. On the undecidability of asynchronous session subtyping. In Javier Esparza and Andrzej S. Murawski, editors, *FOSSACS*, volume 10203 of *LNCS*, pages 441–457, 2017.

[Mil91] Robin Milner. The polyadic π-calculus: a tutorial. In Friedrich L. Bauer, Wilfried Brauer, and Helmut Schwichtenberg, editors, *Logic and Algebra of Specification*, NATO ASI. Springer, 1991.

[MP17a] Claudio Antares Mezzina and Jorge A. Pérez. Causally consistent reversible choreographies: a monitors-as-memories approach. In *PPDP*, pages 127–138. ACM Press, 2017.

[MP17b] Claudio Antares Mezzina and Jorge A. Pérez. Reversibility in session-based concurrency: A fresh look. *Journal of Logic and Algebraic Methods in Programming*, 90:2–30, 2017.

[MYH09] Dimitris Mostrous, Nobuko Yoshida, and Kohei Honda. Global principal typing in partially commutative asynchronous sessions. In Giuseppe Castagna, editor, *ESOP*, volume 5502 of *LNCS*, pages 316–332. Springer, 2009.

[NPW81] Mogens Nielsen, Gordon Plotkin, and Glynn Winskel. Petri nets, event structures and domains, part I. *Theoretical Computer Science*, 13(1):85–108, 1981.

[NY17] Rumyana Neykova and Nobuko Yoshida. Let it recover: multiparty protocol-induced recovery. In Peng Wu and Sebastian Hack, editors, *CC*, pages 98–108. ACM Press, 2017.

[Old86] Ernst-Rüdiger Olderog. TCSP: theory of communicating sequential processes. In Wilfried Brauer, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Advances in Petri Nets*, volume 255 of *LNCS*, pages 441–465. Springer, 1986.

[Pad15] Luca Padovani. Type Reconstruction for the Linear π-Calculus with Composite Regular Types. *Logical Methods in Computer Science*, 11(4), 2015.

[PCPT14] Jorge A. Pérez, Luís Caires, Frank Pfenning, and Bernardo Toninho. Linear logical relations and observational equivalences for session-based concurrency. *Information and Computation*, 239:254–302, 2014.

[Pie02] Benjamin C. Pierce. *Types and Programming Languages*. MIT Press, 2002.

[PS96] Benjamin C. Pierce and Davide Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):376–385, 1996.

[PU15] Iain C.C. Phillips and Irek Ulidowski. Reversibility and asymmetric conflict in event structures. *Journal of Logical and Algebraic Methods in Programming*, 84(6):781 – 805, 2015.

[SDC19] Paula Severi and Mariangiola Dezani-Ciancaglini. Observational Equivalence for Multiparty Sessions. *Fundamenta Informaticae*, 167:267–305, 2019.

[TCP11] Bernardo Toninho, Luís Caires, and Frank Pfenning. Dependent session types via intuitionistic linear type theory. In Peter Schneider-Kamp and Michael Hanus, editors, *PPDP*, pages 161–172. ACM Press, 2011.

[TG18] Emilio Tuosto and Roberto Guanciale. Semantics of global view of choreographies. *Journal of Logic and Algebraic Methods in Programming*, 95:17–40, 2018.

[THK94] Kaku Takeuchi, Kohei Honda, and Makoto Kubo. An interaction-based language and its typing system. In Chris Hankin, editor, *PARLE*, volume 817 of *LNCS*, pages 122–138. Springer, 1994.

[TY14] Francesco Tiezzi and Nobuko Yoshida. Towards reversible sessions. In Alastair F. Donaldson and Vasco T. Vasconcelos, editors, *PLACES*, volume 155 of *EPTCS*, pages 17–24. Open Publishing Association, 2014.

[TY16] Francesco Tiezzi and Nobuko Yoshida. Reversing single sessions. In Simon J. Devitt and Ivan Lanese, editors, *RC*, volume 9720 of *LNCS*, pages 52–69. Springer, 2016.

[VY10] Daniele Varacca and Nobuko Yoshida. Typed event structures and the linear π-calculus. *Theoretical Computer Science*, 411(19):1949–1973, 2010.

[Wad14] Philip Wadler. Propositions as sessions. *Journal of Functional Programming*, 24(2-3):384–418, 2014.

[Win80] Glynn Winskel. *Events in Computation*. PhD thesis, University of Edinburgh, 1980.

[Win82] Glynn Winskel. Event structure semantics for CCS and related languages. In Mogens Nielsen and Erik Meineche Schmidt, editors, *ICALP*, volume 140 of *LNCS*, pages 561–576. Springer, 1982.

[Win88] Glynn Winskel. An introduction to event structures. In Jaco W. de Bakker, Willem P. de Roever, and Grzegorz Rozenberg, editors, *REX: Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, volume 354 of *LNCS*, pages 364–397. Springer, 1988.