A PO Characterisation of Reconfiguration[⋆]

Yehia Abd Alrahman, Mauricio Martel, and Nir Piterman

University of Gothenburg, Gothenburg, Sweden
{yehia.abd.alrahman,nir.piterman,mauricio.martel}@gu.se

Abstract. We consider partial order semantics of concurrent systems in which local reconfigurations may have global side effects. That is, local changes happening to an entity may block or unblock events relating to others, namely, events in which the entity does *not* participate. We show that partial order computations need to capture additional restrictions about event ordering, i.e., restrictions that arise from such reconfigurations. This introduces ambiguity where different partial orders represent exactly the same events with the same participants happening in different orders, thus defeating the purpose of using partial order semantics. To remove this ambiguity, we suggest an extension of partial orders called *glued partial orders*. We show that glued partial orders capture all possible forced reordering arising from said reconfigurations. Furthermore, we show that computations belonging to different glued partial orders are only different due to non-determinism. We consider channeled transition systems and Petri-nets with inhibiting arcs as examples.

1 Introduction

The most common way to represent computations is by considering linear sequences of events. When reasoning about concurrent systems, the linear order semantics of computation does not capture important information about participation in events and the interdependence of events. In order to capture this extra information in computations, instead of linear order, partial order semantics needs to be used. Existing approaches to partial order semantics (cf. *Process semantics* of Petri nets [27,23,32] and *Mazurkiewicz traces* of Zielonka automata [34,17,22]) proved useful in recovering information about the participants of events and (in)dependence of concurrent events.

In this paper, we are interested in concurrenct systems where events are affected by changes happening to non-participants. This situation, which we call in general *reconfiguration*, arises in two types of very different models of concucrrent systems: *Channeled Transition Systems* (CTS) [5,4][1] and *Petri net with inhibitor arcs* (PTI-nets) [21,16,11]. In the first, processes connect and disconnect to channels during execution and by doing so disable and enable communications

[⋆] This work is funded by ERC consolidator grant D-SynMA (No. 772459) and Swedish research council grants: SynTM (No. 2020-03401) and grant (No. 2020-04963).

[1] CTS can be considered as a generalisation of Zielonka automata, supporting rich interactions alongside change of communication interfaces.

on these channels in which they ultimately do not participate. In the second, tokens enter and exit places that inhibit transitions and by doing so disable and enable said transitions without participating in the transitions themselves. In these two settings, dependencies among events emerge dynamically as side-effects of interaction, leading to difficulties in capturing these emergencies in partial-order semantics.

It is possible to suggest a partial order semantics of such types of systems. Indeed, we give a partial order semantics to both types of systems (cf. [20] for an alternative partial order semantics of PTI-nets). However, we recognise that reconfiguration induces another dimension of nondeterminism in these systems. Reconfiguration creates a situation where some events must be ordered with respect to *sequences of other events* dynamically during execution, and thus forcing interleaving in a non-trivial way. That is, from the point of an event, a sequence of other events is considered as a single block and can only happen before or after it.[2] Just like multiple linear sequences correspond to different interleavings arising from the same partial order, reconfigurations lead to multiple partial order computations corresponding to exactly the same events with exactly the same participants happening along a computation just in different orders.

To resolve that, we propose an extension of partial orders with additional objects called glue. Using these structures we can define semantics that characterises reconfiguration. Both reconfiguration points and their corresponding scheduling decisions are captured in a single structure, while preserving a *true*-concurrent execution of independent events.

Contributions. We show how to give partial order semantics to these two types of systems in a way that captures reconfigurations. We use a specialised version of partial orders, that we call *labelled partial orders* (LPO for short). We show how to construct an LPO representing the computations of a specific system. Such LPOs consider only the local views of individual processes / indistinguishable tokens and their interaction information. An LPO captures participation in events and the relations between events. In the spirit of Mazurkiewicz traces, the states of different processes / distinguishable tokens are (strictly) incomparable, that is there is no notion of a global state. This way we can easily single out finite sequences of computation steps where a process (or a group of processes) / tokens execute independently. We can also distinguish individual events from joint ones. As mentioned, despite the fact that an LPO may refer to reconfiguration points, it cannot fully characterise reconfiguration in a single structure. For this reason, we introduce *glued labeled partial orders* (g-LPO, for short), that is an extension of LPO with *glue* to separate a non-deterministic choice from forced scheduling due to reconfiguration. We show that a g-LPO is sufficient to

[2] Note that reconfiguration is an internal event, and is totally hidden from the perspective of an external observer [26] who may only observe message-/token-passing. Indeed, messages or tokens can only indicate the occurrence of exchange but cannot help with noticing that a reconfiguration has happened and what are the consequences of reconfiguration.

represent LPO computations that differ in scheduling due to reconfiguration. We also show that LPO computations belonging to different g-LPO(s) are different due to nondeterministic selection of independent events.

The paper is organised as follows: In Sect. 2, we informally present our partial order semantics and in Sect. 3, we introduce the necessary background. In Sect. 4, we provide the LPO semantics and in Sect. 5 we define the glued partial orders. In Sect. 6 we prove important results on g-LPO with respect to reconfiguration and nondeterminism. In Sect. 7 we present concluding remarks. All proofs are included in the appendix.

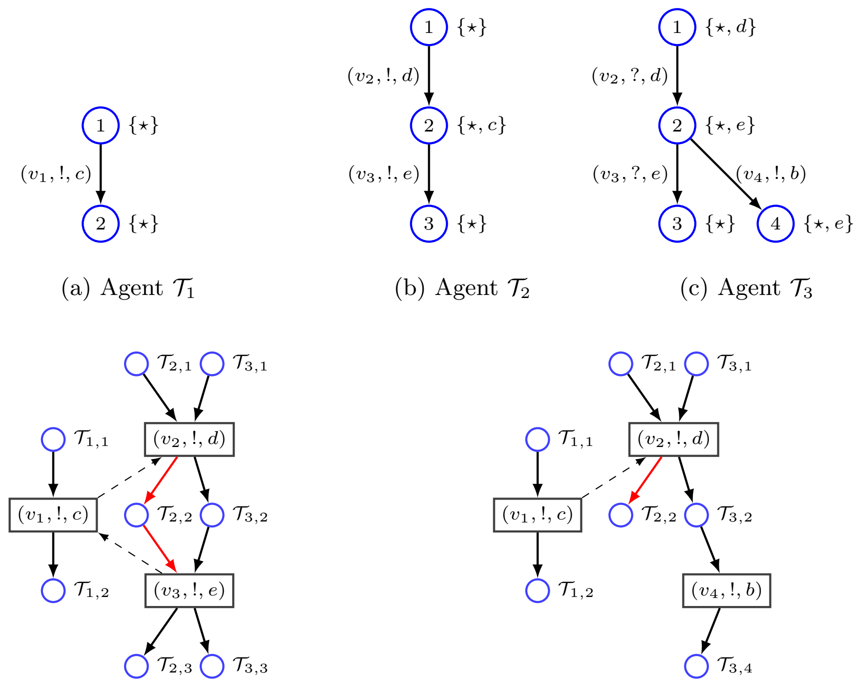
2 Labelled Partial Order Computations in a Nutshell

We use the CTS formalism to informally illustrate the LPO semantics under reconfiguration and the idea behind g-LPO. The example is kept simple to aid the reader, but our semantics can handle much more intricate cases where dependencies are nontrivial.

We consider the CTSs in Fig. 1(a-c) where each CTS $\mathcal{T}_i$ for $i \in \{1, 2, 3\}$ represents an individual agent and their parallel composition defines the system behaviour as we will explain shortly. A CTS $\mathcal{T}_i$ consists of a set of states and transitions. We will use the notation $\mathcal{T}_{i,k}$ to denote that agent $\mathcal{T}_i$ is currently in state k. A state is labelled with a dynamic listening function to define the set of channels that the agent is connected to, including a special nonblocking broadcast channel $\star$. All other channels are blocking multicast. An agent cannot disconnect from the broadcast channel. Each transition is labelled with a message of the form (v, r, ch) where v is the message contents, $r \in \{!, ?\}$ is the role of the transition either send ! or receive ?, and ch is a channel name.

The system behaviour is defined as follows: if there exists an agent with a send transition on a specific channel then for all other agents: In case of broadcast: the sender cannot be blocked and all agents who can supply a matching receive transition participate. In case of multicast: all agents who are listening to the channel must participate by supplying their matching receive transition or otherwise the sender is blocked. For instance, agent $\mathcal{T}_2$ can initially (i.e., in state $\mathcal{T}_{2,1}$) send the message $(v_2, !, d)$ and move to $\mathcal{T}_{2,2}$ and only agent $\mathcal{T}_3$ is initially listening to channel d in $\mathcal{T}_{3,1}$. Thus, $\mathcal{T}_3$ participates (and moves from $\mathcal{T}_{3,1}$ to $\mathcal{T}_{3,2}$) while $\mathcal{T}_1$ stays still as it cannot observe the communication. After the joint transition $\mathcal{T}_2$ starts listening to c (in $\mathcal{T}_{2,2}$) while $\mathcal{T}_3$ disconnects from d and starts listening to e (in $\mathcal{T}_{2,2}$).

Agents can reconfigure their interaction interfaces by updating their listening functions as in the previous example. The side effects of such reconfiguration may change the ordering of events at system level even though the reconfiguration happened internally. For instance, after sending message $(v_2, !, d)$, agent $\mathcal{T}_2$ starts listening to channel c (in state $\mathcal{T}_{2,2}$) but cannot supply a receive transition for this channel. Thus, agent $\mathcal{T}_1$ is now blocked until $\mathcal{T}_2$ exits state $\mathcal{T}_{2,2}$ and disconnects from c. That is, if $(v_2, !, d)$ happened before $(v_1, !, c)$ then $(v_1, !, c)$ may only happen after $(v_3, !, e)$. In other words, $(v_1, !, c)$ is now ordered with respect to the



(a) Agent $\mathcal{T}_1$ (b) Agent $\mathcal{T}_2$ (c) Agent $\mathcal{T}_3$

(d) Three LPO computations and two g-LPO computations

Fig. 1. Channel Transition System CTS

sequence $(v_2, !, d)$,$(v_3, !, e)$. It should be noted that initially (from $\mathcal{T}_{1,1}$ and $\mathcal{T}_{2,1}$) there were no dependencies between $(v_1, !, c)$ and $(v_2, !, d)$, but such dependencies arose as side effects of internal reconfiguration of agent $\mathcal{T}_2$.

Moreover, agent $\mathcal{T}_3$ (from $\mathcal{T}_{3,2}$) may inhibit the sending of message $(v_3, !, e)$ by nondeterministically choosing to send $(v_4, !, b)$ instead and moving to state $\mathcal{T}_{3,4}$. Note that $\mathcal{T}_3$ still listens to e (in $\mathcal{T}_{3,4}$), but cannot supply a matching receive transition, and thus permanently blocks $\mathcal{T}_2$ (in $\mathcal{T}_{2,2}$).

By restricting attention to the interleavings, we have that $(v_1, !, c)$ considers both $(v_2, !, d)$ and $(v_3, !, e)$ as a single block, and their execution cannot be interrupted. Namely, the only viable interleavings (in case $(v_3, !, e)$ is scheduled later) are $(v_1, !, c)$, $(v_2, !, d)$, $(v_3, !, e)$ or $(v_2, !, d)$, $(v_3, !, e)$, $(v_1, !, c)$. Note that this is only from the point of view of $(v_1, !, c)$ and has no implications for other messages. This creates a forced interleaving in a non-trivial way due to the occurrence of non-observable reconfigurations that we cannot reason about from a global perspective. Furthermore, these dependencies among events emerge dynamically as side-effects of interaction, and thus put the correctness of partial order semantics at stake.

To handle this issue, we introduce a partial order semantics of computations under reconfiguration. We illustrate our LPO and g-LPO semantics in Fig. 1(d), which characterises all possible (maximal) computations of the composed system. Here, we use the dashed arrow $\dashrightarrow$ to indicate a happen before relation (or an interleaving order $\rightarrow_i$ as we will see later).

The two diagrams succinctly encode *three* possible LPOs: (i) the LPO obtained from Fig. 1(d) left structure with the dashed arrow from $(v_1, !, c)$ to $(v_2, !, d)$; (ii) the LPO obtained from Fig. 1(d) left structure with the dashed arrow from $(v_3, !, e)$ to $(v_1, !, c)$; and (iii) the LPO obtained from Fig. 1(d) right structure with the dashed arrow from $(v_1, !, c)$ to $(v_2, !, d)$. LPOs (i) and (ii) agree that agent $\mathcal{T}_3$ (in state $\mathcal{T}_{3,2}$) nondeterministically chooses to send $(v_3, !, e)$ while in (iii) $\mathcal{T}_3$ nondeterministically chooses $(v_4, !, b)$. All LPOs capture information about interaction and interdependence among events. Indeed, in all cases we see that both states $\mathcal{T}_{2,1}$ and $\mathcal{T}_{3,1}$ synchronise through the transition $(v_2, !, d)$. States that are not strictly ordered with respect to a common transition are considered concurrent (or unordered). Thus, as in Mazurkiewicz traces there is no notion of a global state. Notice that LPOs (i) and (ii) differ only in the forced interleaving of $(v_1, !, c)$ with respect to the block $(v_2, !, d), (v_3, !, e)$.

Note that both LPOs (i) and (ii) have information both on reconfiguration and nondeterminism, but each individually cannot be used to distinguish the hidden reconfiguration. In fact, $(v_1, !, c) \dashrightarrow (v_2, !, d)$ in (i) indicates that $(v_1, !, c)$ happened before a reconfiguration caused by $(v_2, !, d)$, and $(v_3, !, e) \dashrightarrow (v_1, !, c)$ in (ii) indicates that $(v_1, !, c)$ happened after the reconfiguration. In (iii), due to the different nondeterminsitic choice, the only possible case we have to consider is that of $(v_1, !, c)$ happening before $(v_2, !, d)$.

This suggests that we can actually isolate reconfiguration from nondeterminism by using a more sophisticated structure than LPO, and thus expose the difference in a way that allows reasoning about these hidden events from a global perspective. For this reason, we define g-LPO computations, that are an extension of LPO with a notion of *glue*.

In this simple example, a g-LPO simply drops strict ordering of events with respect to each other (like $(v_1, !, c) \dashrightarrow (v_2, !, d)$ or $(v_3, !, e) \dashrightarrow (v_1, !, c)$), and instead assigns each event a (possibly empty) glue relation defining the *glued* elements from the point of view of that event. The glue relation is defined based on reconfiguration points in CTS, and on inhibitor arcs in Petri nets.

Consider now the structures in Fig. 1(d) without the dashed arrows and, now, with an explanation of the red arrows. These two structures are each a g-LPO. For the one on the left, since $\mathcal{T}_{2,2}$ inhibits $(v_1, !, c)$ all existing incoming and outgoing edges from $\mathcal{T}_{2,2}$ are glued to $\mathcal{T}_{2,2}$. Thus, $(v_1, !, c)$'s glue relation includes these edges (in red). All other transitions have empty glue relations because they are not inhibited. As they are not inhibited, their interdependence is well-captured statically based on their communication. Note that the glue relation is not required to be transitive and the glue only relates states and transitions. In the structure on the right of the figure, $(v_2, !, d)$ is glued only to $\mathcal{T}_{3,2}$. As $(v_4, !, b)$ is scheduled rather than $(v_3, !, e)$, then $\mathcal{T}_{2,2}$ remains as a maximal element.

As we show later, a single g-LPO can be used to characterise reconfiguration and separate it from other sources of nondeterminism in the system.

3 Preliminaries

3.1 Partial Orders and Labeled Partial Orders

We use a specialised form of partial orders to represent computations.

A *partial order* (PO, for short) is a binary relation $\leq$ over a set O that is reflexive, antisymmetric, and transitive. We use $a < b$ for $a \leq b$ and $a \neq b$. We use $a \# b$ for $a \not\leq b$ and $b \not\leq a$, i.e., a and b are incomparable.

A *labelled partial order* (LPO, for short) is $(O, \rightarrow_c, \rightarrow_i, \Sigma, \Upsilon, L)$, where $O = V \uplus E$ is a set of elements partitioned to nodes and edges, respectively, $\rightarrow_c$ and $\rightarrow_i$ are disjoint, anti-reflexive, anti-symmetric, and non-transitive *communication* and *interleaving* order relations over O. We have $\rightarrow_c \subseteq V \times E \cup E \times V$ and $\rightarrow_i \subseteq E \times E$. When $\rightarrow_i = \emptyset$ we omit it from the tuple. We denote $\rightarrow = \rightarrow_c \cup \rightarrow_i$. The relation $\leq$ is the reflexive and transitive closure of $\rightarrow$. We require that $\leq$ is a partial order. The labelling function $L : O \rightarrow \Sigma \cup \Upsilon$ satisfies $L(V) \subseteq \Sigma$ and $L(E) \subseteq \Upsilon$, where Σ is a node alphabet and Υ is an edge alphabet. Given an element $a \in O$ we write ${}^\bullet a$ for $\{b \mid b \rightarrow a\}$ and $a^\bullet$ for $\{b \mid a \rightarrow b\}$.

Intuitively, for CTS, elements in V relate to execution histories of individual agents and elements in E to communication events. Thus, a history belongs to an individual agent and a transition corresponds to either an individual computational step or a synchronisation point among multiple agents. The relation $\rightarrow_c$ captures participation in communication and the relation $\rightarrow_i$ captures order requirements.

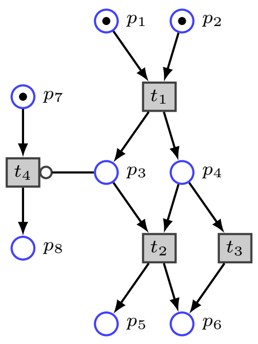
For PTI-nets, elements in V correspond to a history of a token or multiple tokens with the same history and elements in E correspond to transitions. Similarly, $\rightarrow_c$ captures participation in transitions and $\rightarrow_i$ captures order.

3.2 Channelled Transition Systems (CTS)

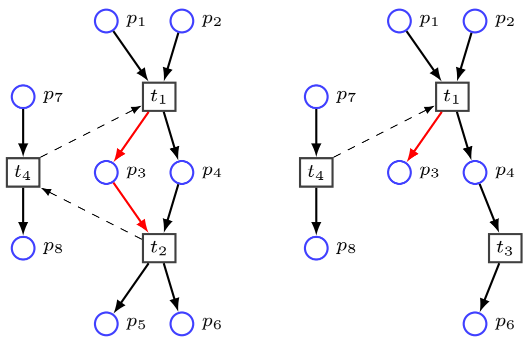
We present Channeled Transition Systems [5,4]. A *Channelled Transition System* (CTS) is a tuple of the form $\mathcal{T} = \langle C, \Lambda, B, S, S_0, R, L, \text{LS}\rangle$, where C is a set of channels, including the broadcast channel ($\star$), Λ is a *state alphabet*, B is a *transition alphabet*, S is a set of states, $S_0 \subseteq S$ is a set of initial states, $R \subseteq S \times B \times S$ is a transition relation, $L : S \rightarrow \Lambda$ is a labelling function, and $\text{LS} : S \rightarrow 2^C$ is a channel-listening function such that for every $s \in S$ we have $\star \in \text{LS}(s)$. That is, a CTS is listening to the broadcast channel in every state. We assume that $B = B^+ \times \{!, ?\} \times C$, for some set B^+. That is, every transition labeled with some $b \in B$ is either a message send (!) or a message receive (?) on some channel $c \in C$. We write $B^!$ for $B^+ \times \{!\} \times C$ and $B^?$ for $B^+ \times \{?\} \times C$.

Given $(b^+, !, c) \in B$ we write $?(b^+, !, c)$ for $(b^+, ?, c)$ and $ch(b^+, -, c)$ for c. That is, $?(b)$ is the corresponding receive transition of a send transition b and $ch(b)$ is the channel of b.

For a receive transition $b = (b^+, ?, c)$ and a state $s \in S$ we write $s \rightarrow_b$ if $c \in \text{LS}(s)$ and there is some s' such that $(s, b, s') \in R$. That is, s is listening on channel c and can participate, i.e., has an outgoing receive transition for b. We



(a) Petri net with Inhibitor arcs



(b) Two possible g-LPO computations

Fig. 2. Petri net with inhibitor arcs

write $s \not\to_b$ if $c \in \text{LS}(s)$ and it is not the case that $s \to_b$. That is, s is listening on channel c and is not able to participate.

A *history* $h = s_0, \ldots, s_n$ is a finite sequence of states such that $s_0 \in S_0$ and for every $0 \leq i < n$ we have that $(s_i, b_i, s_{i+1}) \in R$ for some $b_i \in B$. The length of h is $n+1$, denoted $|h|$. For convenience we generalise notations applying to states to apply to histories. For example, we write $c \in \text{LS}(h)$ when $c \in \text{LS}(s_n)$, $h \to_b$ when $s_n \to_b$ and $h \not\to_b$ for $s_n \not\to_b$. Similarly, if $h = s_0, \ldots, s_n$ and $h' = s_0, \ldots, s_n, s_{n+1}$ where $(s_n, b_n, s_{i+1}) \in R$, we write $(h, b_n, h') \in R$. Let $\textbf{hist}(\mathcal{T})$ be the set of all histories of $\mathcal{T}$.

Consider a system $\mathcal{S} = \mathcal{T}_1 \parallel \cdots \parallel \mathcal{T}_n$ with n CTSs, where $\mathcal{T}_i = \langle C_i, \Lambda_i, B_i, S_i, S_0^i, R_i, L_i, \text{LS}_i \rangle$ is a CTS. We denote $C = \bigcup_i C_i$, and $B = \bigcup_i B_i$ and $B^! = \bigcup_i B_i^!$. A global state of $\mathcal{S}$ is $S = \prod_i S_i$ and $S^0 = \prod_i S_0^i$ is the set of initial states. The global linear order transition relation $\Delta \subseteq S \times B^! \times S$ is defined as follows:

$$\Delta = \left\{ \begin{pmatrix} (s_1, \ldots, s_n), \\ (\upsilon, !, c), \\ (s'_1, \ldots, s'_n) \end{pmatrix} \middle| \begin{array}{l} \exists i \;.\; (s_i, (\upsilon, !, c), s'_i) \in R_i \text{ and } \forall j \neq i \;. \\ \quad (1)\ (s_j, (\upsilon, ?, c), s'_j) \in R_j \text{ and } c \in \text{LS}_j(s_j) \text{ or} \\ \quad (2)\ c \notin \text{LS}_j(s_j) \text{ and } s'_j = s_j \text{ or} \\ \quad (3)\ c = \star, s'_j = s_j, \text{ and } \forall s'' \;.\; (s_j, (\upsilon, ?, c), s'') \notin R_j \end{array} \right\}$$

Intuitively, there exists one sender and potentially multiple receivers. Multicast channels are blocking, i.e., (1) all agents who are listening to the channel must be able to participate in the communication in order for a send to be possible; (2) agents that are not listening ignore the message. The broadcast channel is non-blocking and agents always listen to it, i.e., (1) if they can participate, each supplies a receive transition and receives the message; (3) if they cannot participate in a communication it still goes on without them.

3.3 Petri Nets with Inhibitor Arcs (PTI-nets)

We present Petri Nets with inhibitor arcs [21,16,11]. A Petri net N with inhibitor arcs is a bipartite directed graph $N = \langle P, T, F, I \rangle$, where P and T are the set of places and transitions such that $P \cap T = \emptyset$, $F : (P \times T) \cup (T \times P) \to \mathbb{N}$ is the flow relation, and $I \subseteq (P \times T)$ is the inhibiting relation. We write $(s, s') \in F$ for $F(s, s') > 0$. Without loss of generality, we restrict attention to Petri nets where all transitions have a non-empty preset and a non-empty post-set.

The configuration of a Petri net at a time instant is defined by a *marking.* Formally, let N be a Petri net with a set of places $P = \{p_1, \ldots, p_k\}$. A marking is a function $m : P \to \mathbb{N}$, where $m(p_i)$ corresponds to the number of tokens in p_i, for $i = 1, \ldots, k$. Functions can be added, subtracted, and compared in the usual way. We assume some initial marking m_0. For $p \in P$ let $\boldsymbol{p}$ be the function $\boldsymbol{p} : P \to \{0, 1\}$ such that $\boldsymbol{p}(p) = 1$ and $\boldsymbol{p}(p') = 0$ for every $p' \neq p$. Let M denote the set of all markings.

For a transition $t \in T$ we define the *pre-function* of t, denoted by ${}^\bullet t$, to be ${}^\bullet t(p) = F(p_i, t)$. Similarly, the *post-function* of t is $t^\bullet = F(t, p_i)$.

An inhibitor arc from a place to a transition means that the transition can only fire if no token is on that place. The inhibitor set of a transition t is the set ${}^\circ t = \{p \in P \mid (p, t) \in I\}$, and represents the places to be "tested for absence" of tokens. That is, an inhibiting place allows to prevent the transition firing.

A transition t is enabled at m if for every p we have $m(p) \geq F(p, t)$ and all inhibitor places are empty, i.e., for every $p \in {}^\circ t$ we have $m(p) = 0$. Note that if for some t and $p \in {}^\circ t$ we have $(p, t) \in F$ then t can never fire, i.e., it is blocked.

The running example and the corresponding LPOs and g-LPOs can be modelled in PTI-Nets as in Fig. 2 where the multiplicities for all edges is 1. Intuitively, the inhibitor arc plays the role of a CTS state that listens to a message but does not supply a receive transition.[3]

4 LPO Semantics

In this section, we provide CTSs and PTI-nets with a labelled partial order semantics. The labelled partial order semantics of CTSs is novel while the one of Petri nets extends occurrence nets [23] with event-to-event connections that allow to capture reconfigurations. We include in Appendix C the labelled partial order semantics of asynchronous automata, which do not require the interleaving relation $\to_i$, and, thus, show that the separation of results in this paper only make sense in reconfigurable systems.

4.1 Channelled Transition Systems (CTS)

Consider a system $\mathcal{S} = \mathcal{T}_1 \parallel \cdots \parallel \mathcal{T}_n$, where $\mathcal{T}_i = \langle C_i, \Lambda_i, B_i, S_i, S_0^i, R_i, L_i, \text{LS}_i \rangle$. We denote $C = \bigcup_i C_i$, and $B = \bigcup_i B_i$.

Definition 1 (LPO-computation). *A* computation *of* $\mathcal{S}$ *is an* LPO $(O, \to_c, \to_i, \Sigma, \Upsilon, L)$*, where* $V \subseteq \bigcup_i \boldsymbol{hist}(\mathcal{T}_i)$*,* $\Sigma = V$*,* $\to_c = \to_s \uplus \to_r$ *is the disjoint union of the send and receive relations,* $\Upsilon = \{(\upsilon, !, c) \in B\}$ *are the set of message sends, and for* $h \in V$ *we have* $L(h) = h$*. In addition we require the following:*

[3] A general translation of CTS to PTI-nets is quite involved and loses the distinction between channels and processes.

C1. The edge e_ϵ such that $L(e_\epsilon) = (b, !, \star)$ is the unique minimal element according to $\leq$. For every i, we have $s_i^0 \in V$ and $e_\epsilon \rightarrow_r s_i^0$.

C2. If $h \in V \cap \boldsymbol{hist}(\mathcal{T}_i)$ there is a unique $e \in E$ such that $e \rightarrow_c h$. If $|h| > 1$, there is also a unique $h' \in V \cap \boldsymbol{hist}(\mathcal{T}_i)$ such that $h' \rightarrow_c e$ and either $(h', L(e), h) \in R_i$ or $(h', ?(L(e)), h) \in R_i$.

C3. For every $h \in V$ there is at most one $e \in E$ such that $h \rightarrow_c e$.
That is, h participates in at most one communication.

C4. For every $e \in E \setminus \{e_\epsilon\}$ there is $I \subseteq [n]$ such that all the following hold:

(a) For every $i \in I$ we have $|{}^\bullet e \cap \boldsymbol{hist}(\mathcal{T}_i)| = 1$ and $|e^\bullet \cap \boldsymbol{hist}(\mathcal{T}_i)| = 1$.
That is, for each agent that participates in a communication the edge connects exactly to one predecessor history and one successor history.

(b) There is a unique $i \in I$ and $h, h' \in V \cap \boldsymbol{hist}(\mathcal{T}_i)$ such that $(h, L(e), h') \in R_i$ and $h \rightarrow_s e \rightarrow_s h'$ and for every $i' \in I \setminus \{i\}$ there are $h'', h''' \in V \cap \boldsymbol{hist}(\mathcal{T}_{i'})$ such that $(h'', ?(L(e), h''') \in R_{i'}$ and $h'' \rightarrow_r e \rightarrow_r h'''$.
That is, every communication has a unique sender and the rest are receivers. All these connections satisfy the respective agent transitions.

(c) If $L(e) = (\upsilon, !, c)$ for $c \neq \star$ then for every $h \in V$ such that $c \in \text{LS}(h)$ we have $h \leq e$ or $e \leq h$.
That is, a communication on a multicast channel is ordered with respect to every history that listens to the same channel. Thus, the history either participates in the communication or happens before or after it.

(d) If $L(e) = (\upsilon, !, \star)$ then for every $h \in V$ such that $h \rightarrow_{?(L(e))}$ we have $h \leq e$ or $e \leq h$.
That is, a communication on the broadcast channel is ordered with respect to every history that could participate in the communication.

C5. For every $e \neq e'$ such that $ch(e) = ch(e')$ we have $e \leq e'$ or $e' \leq e$.
That is, all communications on the same channel are ordered.

C6. If $e \rightarrow_i e'$ then there is some $h = s_0, \ldots, s_j$ and one of what follows holds:

(a) $ch(e) = ch(e')$.
(b) $L(e') = (\upsilon, !, c)$ for $c \neq \star$, $h \rightarrow_c e$ and $ch(L(e')) \in \text{LS}(h)$.
(c) $L(e) = (\upsilon, !, c)$ for $c \neq \star$, $e' \rightarrow_c h$ and $ch(L(e')) \in \text{LS}(h)$.
(d) $L(e') = (\upsilon, !, \star)$, $h \rightarrow_c e$ and $h \rightarrow_{?(L(e'))}$.
(e) $L(e) = (\upsilon, !, \star)$, $e' \rightarrow_c h$ and $h \rightarrow_{?(L(e))}$.

That is, we only allow connections between two edges in order to capture the ordering in a single channel (a), to capture the order between multi-cast messages and histories that could be listening to them (b,c), or to capture the order between broadcasts and histories that could participate in them (d,e).

That is, a computation starts from a unique broadcast that initiates all the initial states of $\mathcal{T}_i$ for all i (C1). Every history has a unique communication that leads to it and (if it is not the initial state) the communication connects a unique previous history of the same agent according to the transition of the agent (C2). Every history participates in at most one communication (C3). For every transition there exists a set of agents participating in it (C4). Each agent participates in the communication exactly once (C4a), has one sender and all the rest are receivers (C4b), is ordered with respect to all places that could participate in it

(C4c,d). Then, all communications on the same channel are ordered (C5). Interleaving (C4c,d and C5) is captured by interleaving relation. Communications on the same channel can be ordered (C6a). For a multicast, a history h that could participate in the multicast already participated in a communication (C6b), or the communication leading to h happens after the multicast (C6c). For a broadcast, a history h that could participate in the broadcast already participated in a communication (C6d), or the communication leading to h happens after the broadcast (C6e).

Note that an LPO computation relates histories of individual CTSs, and thus allows to draw relations among finite sequences of individual computation steps of one CTS (or a group of CTSs) with respect to others; Furthermore, a CTS is always listening to the broadcast channel, and thus, it becomes mandatory to order broadcast messages that enable/disable participation to each other.

We will use $\mathbf{comp}(\mathcal{S})$ to denote the set of LPO computations of $\mathcal{S}$.

4.2 Petri Nets with Inhibitor Arcs (PTI-nets)

We now define the LPO semantics of PTI-nets. We start with a definition of histories and then use them to define the vertices and edges of an LPO. In Appendix D, we show our developments for PTI-nets through a detailed example.

Definition 2 (History). *We define the set of histories of a net N by induction.*

We define a special transition t_ϵ such that $t_\epsilon{}^\bullet = m_0$. The pair $(\emptyset, t_\epsilon)$ is a t-history. Note that t_ϵ is not a transition in T.

For a place p, let $h = (S, t)$ be a t-history such that $t^\bullet(p) > 0$. Then we have $(h, p, t^\bullet(p))$ is a p-history. That is, given a t-history h ending in transition t, where p is in $t^\bullet$, then the combination of h, p, and the number of tokes that t puts in p form a p-history.

Consider a transition $t \in T$. A t-history is a pair (S, t), where $S = \{(h_1, i_1), \ldots, (h_n, i_n)\}$ is a multiset satisfying the following. For every j we have $h_j = (-, p, c_j)$ is a p-history, where $c_j \geq i_j$ and ${}^\bullet t = \sum_j i_j \cdot \boldsymbol{p_j}$. That is, the t-history identifies the p-histories from which t takes tokens and the number of occurrences of a p-history in the multiset is the number of tokens taken from it.

Let $\boldsymbol{hist}(N)$ be the set of all histories of N partitioned to $\boldsymbol{hist}_p(N)$ and $\boldsymbol{hist}_t(N)$ in the obvious way. Given a t-history $h = (S, t)$ and a p-history h' we write $h(h')$ for the number of appearances of h' in the multiset S.

We define the labelled partial order semantics of a PTI-net as follows.

Definition 3 (LPO-computation). *A* computation *of N is an* LPO $(O, \rightarrow_c, \rightarrow_i, \Sigma, \Upsilon, L)$*, where $V \subseteq \boldsymbol{hist}_p(N)$, $E \subseteq \boldsymbol{hist}_t(N)$, $\Sigma = P$, $\Upsilon = T$, for a p-history $v = (-, p, i)$ we have $L(v) = p$ and for a t-history (S, t) we have $L(e) = t$, and such that:*

N1. The t-history $(\emptyset, t_\epsilon)$ is the unique minimal element according to $\leq$.

N2. For a p-history $v = (e, p, i) \in V$ we have $e \in E$ and e is the unique edge such that $e \rightarrow_c v$.

N3. For a p-history $v = (h, p, i) \in V$, let $e_1, \ldots, e_j$ be the t-histories such that $v \to_c e_j$. Then, for every k we have $e_k(v) > 0$ and $\sum_k e_k(v) \leq i$.
That is, v leads to t-histories that contain it with the multiplicity of v being respected.

N4. For every $e \in E$, where $e = (\{(v_1, i_1), \ldots, (v_n, i_n)\}, t)$, all the following hold:
- *(a) ${}^\bullet e \cap V = \{v_1, \ldots, v_n\}$ and $e^\bullet \cap V = \{(e, p, t^\bullet(p)) \mid t^\bullet(p) > 0\}$.*
  That is, the connections of a t-history respect the structure of the net.
- *(b) For every $v = (h, p, i) \in V$ such that $p \in {}^\circ L(e)$ we have $e \leq v$ or, where $e_1, \ldots, e_j$ are all the edges such that $v \to_c e_k$, we have $\sum_k e_k(v) = i$ and for every k, $e_k < e$.*
  That is, if a place inhibits a transition, then either the transition happens before the place is visited or all the tokens are taken from the place before the transition happens.
- *(c) If $e \to_i e'$ then there is some v such that either (i) $v \to_c e$ and $(L(v), L(e')) \in I$ or (ii) $e' \to_c v$ and $(L(v), L(e)) \in I$.*
  That is, we only allow connection between two t-histories to capture the forced interleaving due to inhibition.

That is, a computation starts from the dummy transition t_ϵ, which establishes the initial marking (N1) Every other transition is a t-history that connects the p-histories that it contains (N3) to those that contain it (N2). If a place inhibits a transition then either the transition happens before a token arrives to the place or after all tokens left that place (N4b). Namely, if p inhibits t then either t happens before the transition putting token in p or after the transitions taking the tokens from p (N4b). This is possible by adding direct interleaving dependencies ($\to_i$) between edges (N4c).

We will use $\mathbf{comp}(N)$ to denote the set of LPO computations of N.

5 Partial Order with Glue

We extend labeled partial orders with *glue*. Intuitively, two elements are glued from the point of view of another element if they both happen either before or after said element.

Definition 4 (Glue). *A* Glue *over a set O and a relation $\to_c \subseteq O \times O$ is a relation $R \subseteq \to_c$.*

Intuitively, a glue relation R over the set O and a relation $\to_c$ defines pairs of elements that are glued together.

Definition 5 (Glued LPO). *A* glued LPO *(g-LPO, for short) is $\textsc{lpg} = (P, \mathcal{G}, \mathcal{E})$, where $P = (O = V \biguplus E, \to_c, \to_i, \Sigma, \Upsilon, L)$ is an LPO, $\mathcal{G} = \{G_1, \ldots, G_k\}$ is a set of Glue relations over O and $\to_c$, and $\mathcal{E} : \Upsilon \hookrightarrow \mathcal{G}$ labels elements in E (with their edge labels) by glue relations.*

Definition 6 (g-LPO-refinement). *An LPO $\textsc{lpo} = (O, \to_c, \to_i, \Sigma, \Upsilon, L)$ where $O = V \biguplus E$ refines a g-LPO $\textsc{lpg} = (P_g, \mathcal{G}, \mathcal{E})$, denoted $\textsc{lpo} \preceq \textsc{lpg}$, where $P_g = (O, \to_c, \to_i^g\ \Sigma, \Upsilon, L)$ if the following conditions hold:*

- *For every $e \in E$ and $(a,b) \in \mathcal{E}(L(e))$ we have $e \leq a$ or $b \leq e$.*
- *$\rightarrow_i^g \subseteq \rightarrow_i$ and $(e,e') \in (\rightarrow_i \setminus \rightarrow_i^g)$ implies $(e',v) \in \mathcal{E}(L(e))$or $(v,e) \in \mathcal{E}(L(e'))$for some v.*

That is, the two share the relation $\rightarrow_c$, the relation $\rightarrow_i^g$ is preserved and extended by extra interleaving to capture the glue. In order to respect the glue, an edge that is glued to a pair (a,b) must happen either before a or after b.

We show now that g-LPOs enable to remove parts of the interleaving order relation for both PTI-nets and CTSs. g-LPOs capture better reconfiguration by combining multiple orderings due to the same reconfiguration in the same g-LPO.

5.1 Glue Computations for CTSs

Consider a system $\mathcal{S} = \mathcal{T}_1 \parallel \cdots \parallel \mathcal{T}_n$, where $\mathcal{T}_i = \langle C_i, \Lambda_i, B_i, S_i, S_0^i, R_i, L_i, \text{LS}_i\rangle$. We denote $C = \bigcup_i C_i$ and $B = \bigcup_i B_i$.

We now define a *g-computation* for CTS. The differences from the definition of LPO (Definition 1) are highlighted with a "$*$" (C4.(c-d) and C6.(b-e) are removed and *C7 is new).

Definition 7 (g-computation). *A* g-computation *of S is a g-LPO $(P, \mathcal{G}, \mathcal{E})$, where $P = (O, \rightarrow_i, \rightarrow_c, \Sigma, \Upsilon, L_V, L_E)$ and V, E, Σ, Υ, and L are as before, $\rightarrow_c = \rightarrow_s \uplus \rightarrow_r$, where:*

C1. The edge e_ϵ such that $L(e_\epsilon) = (b, !, \star)$ is the unique minimal element according to $\leq$. For every i, we have $s_i^0 \in V$ and $e_\epsilon \rightarrow_r s_i^0$.

C2. If $h \in V \cap \textbf{\textit{hist}}(\mathcal{T}_i)$ there is a unique $e \in E$ such that $e \rightarrow_c h$. If $|h| > 1$, there is also a unique $h' \in V$ such that $h' \rightarrow_c e$ and either $(h', L(e), h) \in R_i$ or $(h', ?(L(e)), h) \in R_i$.

C3. For every $h \in V$ there is at most one $e \in E$ such that $h \rightarrow_c e$.

*C4. For every $e \in E \setminus \{e_\epsilon\}$ there is $I \subseteq [n]$ such that all the following hold:

(a) For every $i \in I$ we have $|{}^\bullet e \cap \textbf{\textit{hist}}(\mathcal{T}_i)| = 1$ and $|e^\bullet \cap \textbf{\textit{hist}}(\mathcal{T}_i)| = 1$.

(b) There is a unique $i \in I$ and $h, h' \in V \cap \textbf{\textit{hist}}(\mathcal{T}_i)$ such that $(h, L(e), h') \in R_i$ and $h \rightarrow_s e \rightarrow_s h'$ and for every $i' \in I \setminus \{i\}$ there are $h'', h''' \in V \cap \textbf{\textit{hist}}(\mathcal{T}_{i'})$ such that $h'' \rightarrow_r e \rightarrow_r h'''$ and $(h'', ?(L(e), h''') \in R_{i'}$.

C5. For every $e \neq e'$ such that $ch(e) = ch(e')$ we have $e \leq e'$ or $e' \leq e$.

*C6. If $e \rightarrow_i e'$ then the following holds:

(a) $ch(e) = ch(e')$.

*C7. For every $(v, !, c) \in B$ then

$$\begin{aligned}\mathcal{E}((v,!,c)) \;=\; & \{(h,e) \mid \textit{for } c = \star,\; h \rightarrow_c e \textit{ and } h \rightarrow_{?(v,!,c)}\} \cup \\ & \{(e,h) \mid \textit{for } c = \star, e \rightarrow_c h \textit{ and } h \rightarrow_{?(v,!,c)}\} \cup \\ & \{(h,e) \mid \textit{for } c \neq \star,\; h \rightarrow_c e \textit{ and } c \in \text{LS}(h)\} \cup \\ & \{(e,h) \mid \textit{for } c \neq \star, e \rightarrow_c h \textit{ and } c \in \text{LS}(h)\}\end{aligned}$$

We drop from the interleaving relation all order relations that correspond to reconfiguration and keep only those that correspond to the usage of a common resource. Furthermore, we assign each broadcast and multicast message with a glue relation. That is, for every broadcast b add all *existing* ingoing and outgoing messages of histories that may participate in m. The rationale is that if such histories can participate in a broadcast then they cannot be enabled independently from the broadcast as they would participate in it. For every multicast m add all *existing* ingoing and outgoing messages of histories that either block m or could participate in m. The rationale is that such histories cannot be independent from the multicast as they either block it or would participate in it. Note that $^*C7$ adds one glue for every multicast *channel* but one for every broadcast.

We use $\mathbf{comp}_g(\mathcal{S})$ to denote the set of g-computations of CTS $\mathcal{S}$ and show that it indeed captures the same notion of computation.

Theorem 1. *Given a CTS* $\mathcal{T}$*,* $\boldsymbol{comp}(\mathcal{T}) = \{\pi \mid \pi \preceq \pi_g \wedge \pi_g \in \boldsymbol{comp}_g(\mathcal{T})\}$.

5.2 Glue Computations for PTI-nets

Let $N = \langle P, T, F, I\rangle$ be a PTI-net and m_0 its initial marking. We now define a *g-computation*. The differences from the definition of LPO (Def. 3) are highlighted with a "$*$" (N4.(b-c) are removed and *N5 is new).

Definition 8 (g-computation). *A* g-computation *of N is a g-*LPO $(P, \mathcal{G}, \mathcal{E})$*, where* $P = (O, \rightarrow_c, \Sigma, \Upsilon, L)$*, the components V, E, Σ, Υ, and L are as for* LPO*, and the following holds.*

N1. The t-history $(\emptyset, t_\epsilon)$ is the unique minimal element according to $\leq$.

N2. For a p-history $v = (e, p, i) \in V$ we have $e \in E$ and e is the unique edge such that $e \rightarrow_c v$.

N3. For a p-history $v = (h, p, i) \in V$, let $e_1, \ldots, e_j$ be the t-histories such that $v \rightarrow_c e_j$. Then, for every j we have $e_j(v) > 0$ and $\sum_j e_j(v) \leq i$.
That is, v leads to t-histories that contain it with the number of tokens in v being respected.

**N4. For every $e \in E$, where $e = (\{(v_1, i_1), \ldots, (v_n, i_n)\}, t)$ the following holds:*
**(a)* $^\bullet e = \{v_1, \ldots, v_n\}$ *and* $e^\bullet = \{(e, p, t^\bullet(p)) \mid t^\bullet(p) > 0\}$.

**N5. We define a predicate capturing that a place is left without tokens. For a p-history $v = (h, p, i)$, let $e_1, \ldots, e_j$ be the t-histories such that $v \rightarrow_c e_j$. If $\sum_j e_j(v) = i$ we write $f(v)$. Otherwise, it is the case that $\neg f(v)$.*
For every $t \in T$ we have:

$$\begin{array}{rcll}
\mathcal{E}(t) & = & \{(v,e) \mid v \rightarrow_c e \text{ and } (L(v), t) \in I\} & \cup \\
& & \{(e,v) \mid e \rightarrow_c v \text{ and } (L(v), t) \in I\} & \cup \\
& & \{(v,e) \mid \exists v' \,.\, (L(v'), t) \in I,\ \neg f(v'),\ v' \leq v,\ \text{and } v \rightarrow_c e\} & \cup \\
& & \{(e,v) \mid \exists v' \,.\, (L(v'), t) \in I,\ \neg f(v'), v' \leq e,\ \text{and } e \rightarrow_c v\} &
\end{array}$$

That is, for a transition t, add all existing *ingoing and outgoing transitions of places that inhibit t to t's glue. Moreover, if some place that inhibits t has some tokens left in it, then whatever happens after that place is glued as well.*

That is, we drop $\rightarrow_i$ and assign each inhibited transition with a glue relation.

We use $\mathbf{comp}_g(N)$ to denote the set of g-computations of Petri net N and show that it indeed captures the same notion of computation.

Theorem 2. *Given a PTI-net N,* $\boldsymbol{comp}(N) = \{\pi \mid \pi \preceq \pi_g \wedge \pi_g \in \boldsymbol{comp}_g(N)\}$.

6 Separating Choice and Reconfiguration-Forced Interleaving

We show that g-LPOs distinguish nondeterministic choice, which corresponds to different g-LPOs, and interleaving choices due to reconfiguration, which correspond to different ways to refer to glue. For both CTS and PTI-nets, we show that distinct g-LPOs contain different nondeterministic or order choices. Thus, we manage to define *one* structure that captures all possible interleavings and reconfigurations together.

6.1 Choice vs Interleaving in CTSs

A choice that distinguishes two computations for a CTS is either (a) a situation where all the agents have exactly the same history and at least one agent participates in a different interaction or (b) communications on the same channel are ordered differententl. Note that as channels are global resources, the case that changing the order of communications on a channel does not have side effects is accidental. Indeed, such a change of order could have side effects and constitutes a different choice.

We show that every two distinct g-computations of the same CTS have a joint history of some agent that "sees the difference" or a channel that transfers messages in a different order. Difference for a history is either maximality in one and not the other or extension by different communications in the two g-computations.

Theorem 3. *Given a CTS $\mathcal{T}$ and two different g-*LPO*s $G_1, G_2 \in \boldsymbol{comp}_g(N)$ then one of the following holds:*

1. *For some agent i there exists a history h_i in both G_1 and G_2 such that either h_i is maximal in G_α and not maximal $G_{3-\alpha}$, where $\alpha \in \{1, 2\}$;*
2. *For some agent i there exists a history h_i in both G_1 and G_2 such that for the edges e_1 and e_2 such that $h_i \rightarrow_{c_1} e_1$ and $h_i \rightarrow_{c_2} e_2$ we have $L_1(e_1) \neq L_2(e_2)$;*
3. *or; There is a pair of agents i and i' and histories h_i and $h_{i'}$ in both G_1 and G_2 such that the order between the communications of i and i' is different in G_1 and G_2.*

Theorem 3 is not true for LPOs as shown by the LPOs and g-LPO of the CTS in Figure 1. We note that by the proof of Theorem 1 all the LPOs that disagree only on forced interleavings are refined by the same g-LPO.

6.2 Choice vs Interleaving in PTI-nets

A choice that distinguishes two computations is a situation where a set of tokens have exactly the same history and they do a different exchange. We show that every two distinct g-computations of the same net have a set of tokens that "see the difference". That is, they participate in a different transition in the two g-computations. This includes the option of tokens in one g-computation participating in a transition and tokens in the other g-computation not continuing.

Theorem 4. *Given a Petri net P and two different g-*LPO*s $G_1, G_2 \in \boldsymbol{comp}_g(N)$ then one of the following holds:*

1. *There is a node v_i in both G_1 and G_2 such that the number of tokens not taken from v_i in G_1 and G_2 is different.*
2. *There is a set of p-histories $v_1, \ldots, v_n$ in both G_1 and G_2 that participate in some transition t in G_α but not in $G_{3-\alpha}$, where $\alpha \in \{1, 2\}$.*

Note that item 2 includes the case where the transition t happens in both G_1 and G_2 but takes a different number of tokens from every node. This difference is significant as the nodes communicate via the identified transition and share the knowledge about the difference.

Theorem 4 is not true for LPOs. This is already shown by the very simple examples in Figure 2(b). Indeed, in the two LPOs demonstrated by the dashed arcs in the figure, all sets of nodes participate in exactly the same transitions.

We note that by the proof of Theorem 2 all the LPOs that disagree only on forced interleavings are refined by the same g-LPO.

7 Concluding Remarks

We laid down the basis to reason about systems in which events are affected also by non-participants. We showed how to isolate forced interleaving decisions of the system due to such effects, and other decisions due to standard concurrent execution of independent events. This was shown for CTS [5,4] and PTI-nets [16,11], which cover a wide range of interaction capabilities from two different schools of concurrency. In particular, CTS capture channel communication and require order of events without flow of information (captured through the interleaving relation) while PTI-nets are unbounded and more general. We proposed, for both, a partial order semantics, named LPO, of computations under reconfiguration. An LPO supports event-to-event connections that allows to refer to reconfiguration points. Moreover, to fully characterise reconfiguration in a single structure, we proposed a glued LPO semantics, named g-LPO. The latter is able to fully isolate scheduling decisions due to reconfiguration from the ones due to standard concurrency. We show that any LPO computation is only a refinement of some g-LPO of the same system. Finally, we prove important results on g-LPO with respect to reconfiguration and nondeterminism.

Perspectives and future work: Capturing all possible interleavings in a single structure offers opportunities in terms of specification and verification. For example, using languages of linear sequences as a specification language for concurrent system requires some care. Indeed, languages that include certain interleavings of the same computation and exclude others are obviously inappropriate as specifications: there is no system that satisfies them. Invariability under interleaving *without reconfiguration* is easy to check using some representations of languages (deterministic automata) but harder using other representations (temporal logic). We do not know to characterise languages of linear sequences that capture all possible interleavings with reconfiguration. Thus, creating structures that capture precisely such behaviour is important for the definition of appropriate specification languages. Studying g-LPOs could give us insights into the properties of languages of linear sequences that are appropriate to specify concurrent systems with reconfigurations. We could also exploit g-LPO semantics to define specifications over g-LPO computations (rather than linear sequences or LPOs).

Related works: The prevalent approach to semantics of reconfigurable interactions is based on linear order semantics (cf. Pi-calculus [25,15], Mobile Ambients [13], Applied Pi-calculus [1], Psi-calculus [10,8], concurrent constraint programming [30,18], fusion calculus [33], the *AbC* calculus [2,3], ReCiPe [4] etc.). This semantics cannot distinguish the different choices of the system from a global perspective. It hides information about interactions and possible interdependence among events. In fact, linear order semantics ignores the possible concurrency of events, which can be important e.g. for judging the temporal efficiency of the system [32]. Linear order semantics comes even shorter to capture information about reconfiguration from an external observer's point of view.

Partial order semantics (cf. *Process semantics* of Petri nets [27,23,32] and *Mazurkiewicz traces* of Zielonka automata [34,17,22]), on the other hand, is able to refer to the interaction and event dependencies, but does not deal very well with reconfiguration. This is because the latter formalisms have fixed interaction structures, and thus the interdependence of events is defined structurally. Reconfiguration, on the other hand, enforces reordering of events dynamically in non-trivial ways, and thus makes defining correct partial order semantics very challenging. As shown in [20], some aspects of concurrency are almost impossible to tackle in either linear-order or partial-order causality-based models, and one of them is PTI-nets [16]. In fact, reconfiguration increases the expressive power of the formalism, e.g., adding inhibitor arcs to Petri nets makes them Turing Powerful [6]. However, this expressive power does not come without a cost. It prevents most analysis techniques for standard Petri nets [11].

Partial order semantics for PTI-nets are given in [21] and [20]. Much like our LPOs, they represent different forced interleavings *separately.* As they use occurrence nets they have many more ways to represent essentially the same computation due to symmetry between tokens. Relational Structures [20] add an additional "not later than" relation to partial orders. Their emphasis is on providing a general semantic framework for concurrent systems. Thus, relational structures handle issues like priority and error recovery, which we do not handle. However, they are not concerned with uniqueness of representation. So the two works serve different purposes and it would be interesting to investigate mutual extensions.

References

1. Martín Abadi, Bruno Blanchet, and Cédric Fournet. The applied pi calculus: Mobile values, new names, and secure communication. *J. ACM*, 65(1):1:1–1:41, 2018. `doi:10.1145/3127586`.
2. Yehia Abd Alrahman, Rocco De Nicola, and Michele Loreti. A calculus for collective-adaptive systems and its behavioural theory. *Inf. Comput.*, 268, 2019. `doi:10.1016/j.ic.2019.104457`.
3. Yehia Abd Alrahman, Rocco De Nicola, and Michele Loreti. Programming interactions in collective adaptive systems by relying on attribute-based communication. *Sci. Comput. Program.*, 192:102428, 2020. `doi:10.1016/j.scico.2020.102428`.
4. Yehia Abd Alrahman, Giuseppe Perelli, and Nir Piterman. Reconfigurable interaction for MAS modelling. In *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, AAMAS '20, Auckland, New Zealand, May 9-13, 2020*, pages 7–15. International Foundation for Autonomous Agents and Multiagent Systems, 2020.
5. Yehia Abd Alrahman and Nir Piterman. Modelling and verification of reconfigurable multi-agent systems. *Auton. Agents Multi Agent Syst.*, 35(2):47, 2021. `doi:10.1007/s10458-021-09521-x`.
6. Tilak Agerwala. A complete model for representing the coordination of asynchronous processes. Technical report, Johns Hopkins Univ., Baltimore, Md.(USA), 1974.
7. Yehia Abd Alrahman and Hugo Torres Vieira. A coordination protocol language for power grid operation control. *J. Log. Algebraic Methods Program.*, 109, 2019. `doi:10.1016/j.jlamp.2019.100487`.
8. Jesper Bengtson, Magnus Johansson, Joachim Parrow, and Björn Victor. Psi-calculi: a framework for mobile processes with nominal data and logic. *Logical Methods in Computer Science*, 7(1), 2011. `doi:10.2168/LMCS-7(1:11)2011`.
9. Michele Boreale, Rocco De Nicola, and Rosario Pugliese. Basic observables for processes. *Inf. Comput.*, 149(1):77–98, 1999. `doi:10.1006/inco.1998.2755`.
10. Johannes Borgström, Shuqin Huang, Magnus Johansson, Palle Raabjerg, Björn Victor, Johannes Åman Pohjola, and Joachim Parrow. Broadcast psi-calculi with an application to wireless protocols. *Software and System Modeling*, 14(1):201–216, 2015. `doi:10.1007/s10270-013-0375-z`.
11. Nadia Busi. Analysis issues in petri nets with inhibitor arcs. *Theor. Comput. Sci.*, 275(1-2):127–177, 2002. `doi:10.1016/S0304-3975(01)00127-X`.
12. Marco Carbone and Fabrizio Montesi. Deadlock-freedom-by-design: multiparty asynchronous global programming. *ACM SIGPLAN Notices*, 48(1):263–274, 2013.
13. Luca Cardelli and Andrew D. Gordon. Mobile ambients. *Electr. Notes Theor. Comput. Sci.*, 10:198–201, 1997. `doi:10.1016/S1571-0661(05)80699-1`.
14. Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. Strategy logic. *Inf. Comput.*, 208(6):677–693, 2010. `doi:10.1016/j.ic.2009.07.004`.
15. Cristian Ene and Traian Muntean. Expressiveness of point-to-point versus broadcast communications. In *Fundamentals of Computation Theory*, pages 258–268. Springer, 1999.
16. Michael J. Flynn and Tilak Agerwala. Comments on capabilities, limitations and correctness of petri nets. In G. Jack Lipovski and Stephen A. Szygenda, editors, *Proceedings of the 1st Annual Symposium on Computer Architecture, Gainesville, FL, USA, December 1973*, pages 81–86. ACM, 1973. `doi:10.1145/800123.803973`.

17. Blaise Genest, Hugo Gimbert, Anca Muscholl, and Igor Walukiewicz. Optimal zielonka-type construction of deterministic asynchronous automata. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part II*, volume 6199 of *Lecture Notes in Computer Science*, pages 52–63. Springer, 2010. `doi:10.1007/978-3-642-14162-1\_5`.
18. David R. Gilbert and Catuscia Palamidessi. Concurrent constraint programming with process mobility. In *Computational Logic - CL 2000, First International Conference, London, UK, 24-28 July, 2000, Proceedings*, pages 463–477, 2000. `doi:10.1007/3-540-44957-4\_31`.
19. Kohei Honda and Nobuko Yoshida. On reduction-based process semantics. *Theoretical Computer Science*, 151(2):437–486, 1995. 13th Conference on Foundations of Software Technology and Theoretical Computer Science. URL: `https://www.sciencedirect.com/science/article/pii/0304397595000747`, `doi:https://doi.org/10.1016/0304-3975(95)00074-7`.
20. Ryszard Janicki, Jetty Kleijn, Maciej Koutny, and Lukasz Mikulski. Relational structures for concurrent behaviours. *Theor. Comput. Sci.*, 862:174–192, 2021. `doi:10.1016/j.tcs.2020.10.019`.
21. H. C. M. Kleijn and Maciej Koutny. Process semantics of general inhibitor nets. *Inf. Comput.*, 190(1):18–69, 2004. `doi:10.1016/j.ic.2003.11.002`.
22. Siddharth Krishna and Anca Muscholl. A quadratic construction for zielonka automata with acyclic communication structure. *Theor. Comput. Sci.*, 503:109–114, 2013. `doi:10.1016/j.tcs.2013.07.015`.
23. José Meseguer, Ugo Montanari, and Vladimiro Sassone. On the semantics of petri nets. In *International Conference on Concurrency Theory*, pages 286–301. Springer, 1992.
24. Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, I. *Inf. Comput.*, 100(1):1–40, 1992. `doi:10.1016/0890-5401(92)90008-4`.
25. Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, ii. *Information and computation*, 100(1):41–77, 1992.
26. Robin Milner and Davide Sangiorgi. Barbed bisimulation. In W. Kuich, editor, *Automata, Languages and Programming*, pages 685–695, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.
27. Carl Adam Petri and Wolfgang Reisig. Petri net. *Scholarpedia*, 3(4):6477, 2008. `doi:10.4249/scholarpedia.6477`.
28. Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *Conference Record of the Sixteenth Annual ACM Symposium on Principles of Programming Languages, Austin, Texas, USA, January 11-13, 1989*, pages 179–190, 1989. `doi:10.1145/75277.75293`.
29. Davide Sangiorgi and David Walker. *The pi-calculus: a Theory of Mobile Processes.* Cambridge university press, 2003.
30. Vijay A. Saraswat and Martin C. Rinard. Concurrent constraint programming. In *Conference Record of the Seventeenth Annual ACM Symposium on Principles of Programming Languages, San Francisco, California, USA, January 1990*, pages 232–245, 1990. `doi:10.1145/96709.96733`.
31. Alin Stefanescu, Javier Esparza, and Anca Muscholl. Synthesis of distributed algorithms using asynchronous automata. In Roberto M. Amadio and Denis Lugiez, editors, *CONCUR 2003 - Concurrency Theory, 14th International Conference, Marseille, France, September 3-5, 2003, Proceedings*, volume 2761 of *Lec-*

ture Notes in Computer Science, pages 27–41. Springer, 2003. `doi:10.1007/978-3-540-45187-7\_2`.

32. Walter Vogler. Partial order semantics and read arcs. *Theor. Comput. Sci.*, 286(1):33–63, 2002. `doi:10.1016/S0304-3975(01)00234-1`.
33. Lucian Wischik and Philippa Gardner. Explicit fusions. *Theor. Comput. Sci.*, 340(3):606–630, 2005. `doi:10.1016/j.tcs.2005.03.017`.
34. Wieslaw Zielonka. Notes on finite asynchronous automata. *RAIRO Theor. Informatics Appl.*, 21(2):99–135, 1987.

A Proofs for Section 5 (Partial Order with Glue)

Lemma 1. *Given a CTS $\mathcal{T}$ and an* LPO $\pi_1 \in \boldsymbol{comp}(\mathcal{T})$*, there exists a corresponding g-*LPO $\pi_2 \in \boldsymbol{comp}_g(\mathcal{T})$ *such that* $\pi_1 \preceq \pi_2$.

Proof. Let π_2 be the g-LPO obtained from π_1 by using the partial order induced by $\rightarrow_c$ of π_1, by $\rightarrow_i$ of π_1 whenever $e \rightarrow_i e'$ implies $ch(e) = ch(e')$, and adding the glue relations according to Def. 7.

Note that by construction both π_1 and π_2 agree on $\rightarrow_c \subseteq V \times E \cup E \times V$ and only agree on $\rightarrow_i \subseteq E \times E$ whenever $e \rightarrow_i e'$ implies $ch(e) = ch(e')$. We have to show that the conditions of Definition 6 hold.

Consider some $e \in E$ and $(a, b) \in \mathcal{E}(L(e))$. We have to show that $e \leq a$ or $b \leq e$. By definition we know that $a \rightarrow_c b$. We show that either $e \leq a$ or $b \leq e$. We have the following cases.

- $ch(L(e))$ is a multicast channel:
  - If $a \in V$ and $b \in E$ then by definition $ch(L(e)) \in \textsc{ls}(a)$. By $C4(c)$ in definition 1 we have that either $e \leq a$ or $a \leq e$. If $e \leq a$ we are done. If $a \leq e$ then from $a \rightarrow_c b$ it follows that either $e = b$ or $b < e$.
  - If $a \in E$ and $b \in V$ then by definition $ch(L(e)) \in \textsc{ls}(b)$. By $C4(c)$ in definition 1 we have that either $e \leq b$ or $b \leq e$. If $b \leq e$ we are done. If $e \leq b$ then from $a \rightarrow_c b$ it follows that either $e = a$ or $e < a$.
- $ch(L(e))$ is the broadcast channel:
  - If $a \in V$ and $b \in E$ then $a \rightarrow_{?(L(e))}$. By $C4(d)$ in definition 1 we have that either $e \leq a$ or $a \leq e$. If $e \leq a$ we are done. If $a \leq e$ then from $a \rightarrow_c b$ it follows that either $e = b$ or $b < e$.
  - If $a \in E$ and $b \in V$ then $b \rightarrow_{?(L(e))}$. By $C4(d)$ in definition 7 we have that either $e \leq b$ or $b \leq e$. If $b \leq e$ we are done. If $e \leq b$ then from $a \rightarrow_c b$ it follows that either $e = a$ or $e < a$.

Consider $e, e' \in E$ such that $(e, e') \in \rightarrow_i^g$. By construction we have that $(e, e') \in \rightarrow_i$, and thus $\rightarrow_i^g \subseteq \rightarrow_i$. Consider $e, e' \in E$ such that $(e, e') \in (\rightarrow_i \setminus \rightarrow_i^g$). We show that for some v either $(v, e) \in \mathcal{E}(L(e'))$ or $(e', v) \in \mathcal{E}(L(e))$. By $C6(b) - (e)$ in Definiton 1 there exists $v \in V$ such that one of the following holds.

- $ch(L(e')) \neq \star$, $ch(L(e')) \in \textsc{ls}(v)$ and $v \rightarrow_c e$. By ${}^*C7$ in Definition 7, we have that $(v, e) \in \mathcal{E}(L(e'))$ as required.

- $ch(e) \neq \star$, $ch(L(e)) \in \text{LS}(v)$ and $e' \rightarrow_c v$. By ${}^*C7$ in Definition 7, we have that $(e', v) \in \mathcal{E}(L(e))$ as required.
- $ch(e') = \star$, $v \rightarrow_{?(L(e'))}$ and $v \rightarrow_c e$. By ${}^*C7$ in Definition 7, we have that $(v, e) \in \mathcal{E}(L(e'))$ as required.
- $ch(e) = \star$, $v \rightarrow_{?(L(e))}$ and $e' \rightarrow_c v$. By ${}^*C7$ in Definition 7, we have that $(e', v) \in \mathcal{E}(L(e))$ as required.

Lemma 2. *Given a CTS $\mathcal{T}$, a g-*LPO *$\pi_1 \in \boldsymbol{comp}_g(\mathcal{T})$, and an* LPO *$\pi_2$ such that $\pi_2 \preceq \pi_1$ then $\pi_2 \in \boldsymbol{comp}(\mathcal{T})$.*

Proof. Given that $\pi_2 \preceq \pi_1$, it follows that both π_2 and π_2 agree on $\rightarrow_c \subseteq V \times E \cup E \times V$ and only agree on $\rightarrow_i \subseteq E \times E$ whenever $e \rightarrow_i e'$ implies $ch(e) = ch(e')$. Hence, it is sufficient to prove that $C4(c) - (d)$ and $C6(b) - (e)$ in Definition 1 hold for π_2.
We prove $C4(c) - (d)$. Consider some $e \in E$. We have the following cases.

- $ch(L(e))$ is a multicast channel:
  Consider some $v \in V$ such that $ch(L(e)) \in \text{LS}(v)$. We have to show that $e \leq v$ or $v \leq e$. By Definition 7 (${}^*C7$), there is some e' such that one of the following cases holds.
  - $(v, e') \in \mathcal{E}(L(e))$ where $v \rightarrow_c e'$. By refinement, we have that if $(v, e') \in \mathcal{E}(L(e))$ then either $e \leq v$ as required or $e' \leq e$, which implies that $v \leq e$.
  - $(e', v) \in \mathcal{E}(L(e))$ where $e' \rightarrow_c v$. By refinement, we have that if $(e', v) \in \mathcal{E}(L(e))$ then either $v \leq e$ as required or $e \leq e'$, which implies that $e \leq v$.
- $ch(L(e))$ is a broadcast channel:
  Consider some $v \in V$ such that $v \rightarrow_{?(L(e))}$. We have to show that $e \leq v$ or $v \leq e$. By Definition 7 (${}^*C7$), there is some e' such that one of the following cases holds.
  - $(v, e') \in \mathcal{E}(L(e))$ where $v \rightarrow_c e'$. By refinement, we have that if $(v, e') \in \mathcal{E}(L(e))$ then either $e \leq v$ as required or $e' \leq e$, which implies that $e \leq v$.
  - $(e', v) \in \mathcal{E}(L(e))$ where $e' \rightarrow_c v$. By refinement, we have that if $(e', v) \in \mathcal{E}(L(e))$ then either $v \leq e$ as required or $e \leq e'$, which implies that $e \leq v$.

We prove $C6(b) - (e)$. Consider $(e, e') \in \rightarrow_i$ such that $(e, e') \in (\rightarrow_i \setminus \rightarrow_i^g)$. By refinement, we have one of the following cases hold.

- $ch(L(e))$ is a multicast channel:
  - $(e', v) \in \mathcal{E}(L(e))$ for some v. By Definition 7 (${}^*C7$), we have that $e' \rightarrow_c v$ and $ch(L(e)) \in \text{LS}(v)$ as required.
  - $(v, e) \in \mathcal{E}(L(e'))$ for some v. By Definition 7 (${}^*C7$), we have that $v \rightarrow_c e$ and $ch(L(e)) \in \text{LS}(v)$ as required.
- $ch(L(e))$ is a broadcast channel:
  - $(e', v) \in \mathcal{E}(L(e))$ for some v. By Definition 7 (${}^*C6$), we have that $e' \rightarrow_c v$ and $v \rightarrow_{?(L(e))}$ as required.
  - $(v, e) \in \mathcal{E}(L(e'))$ for some v. By Definition 7 (${}^*C6$), we have that $v \rightarrow_c e$ and $v \rightarrow_{?(L(e'))}$ as required.

Theorem 1. *Given a CTS $\mathcal{T}$, $\boldsymbol{comp}(\mathcal{T}) = \{\pi \mid \pi \preceq \pi_g \wedge \pi_g \in \boldsymbol{comp}_g(\mathcal{T})\}$.*

Proof. The proof follows by Lemma 1 and Lemma 2.

Lemma 3. *Given a Petri net N and an* LPO $\pi_1 \in \boldsymbol{comp}(N)$*, there exists a corresponding g*-LPO $\pi_2 \in \boldsymbol{comp}_g(N)$ *such that* $\pi_1 \preceq \pi_2$*.*

Proof. Let π_2 be the g-LPO obtained from π_1 by using $\rightarrow_c$ of π_1, setting $\rightarrow_i^g = \emptyset$, and adding the glue relations according to Def. 8.

We have to show that the conditions of Def. 6 hold. Note that by construction both π_1 and π_2 agree on $\rightarrow_c \subseteq V \times E \cup E \times V$ and only disagree in terms of $\rightarrow_i \subseteq E \times E$ and the glue.

Consider some e such that $L(e) = t \in T$ and $(a,b) \in \mathcal{E}(t)$. We have to show that $e \leq a$ or $b \leq e$. By definition we know that $a \rightarrow_c b$. We have the cases:

– If $a = (h,p,i) \in V$, $b \in E$ and $(L(a),t) \in I$. By N4b in Def. 3 we have that either (i) $e \leq a$ or (ii) $e_1, \dots, e_j$ are all the edges such that $a \rightarrow_c e_k$, we have $\sum_k e_k(a) = i$ and for every k, $e_k < e$. If Case: (i) then we are done. If Case: (ii) then from $a \rightarrow_c b$ it follows that $b < e$. Note that Case: (ii) does not apply if $\neg f(a)$.
– If $a \in E$, $b \in V$ and $(L(b),t) \in I$. By N4b in Def. 3 we have that either (i) $e \leq b$ or (ii) $e_1, \dots, e_j$ are all the edges such that $b \rightarrow_c e_k$, we have $\sum_k e_k(b) = i$ and for every k, $e_k < e$. If Case: (i) then from $a \rightarrow_c b$ it follows that $e < a$. If Case: (ii) then we are done. Note that Case: (ii) does not apply if $\neg f(b)$.
– If $a = (h,p,i) \in V$, $b \in E$, $(L(c),t) \in I$, $c \leq a$ and $\neg f(c)$. By N4b in Def. 3 we have that either (i) $e \leq c$ or (ii) $e_1, \dots, e_j$ are all the edges such that $c \rightarrow_c e_k$, we have $\sum_k e_k(c) = i$ and for every k, $e_k < e$. If Case: (i) then by $c \leq a$ it follows that $e \leq a$. Note that since $\neg f(c)$ there does not exist an LPO such that Case: (ii) holds.
– If $a \in E$, $b \in V$ and $(L(c),t) \in I$, $c \leq a$ and $\neg f(c)$. By N4b in Def. 3 we have that either (i) $e \leq c$ or (ii) $e_1, \dots, e_j$ are all the edges such that $c \rightarrow_c e_k$, we have $\sum_k e_k(c) = i$ and for every k, $e_k < e$. If Case: (i) then by $c \leq a$ it follows that $e \leq a$. Note that since $\neg f(c)$ there does not exist an LPO such that Case: (ii) holds.

Consider some $(e, e') \in \rightarrow_i$. We have to show that either $(e', v) \in \mathcal{E}(L(e))$ for some v or $(v, e) \in \mathcal{E}(L(e'))$. By definition, we have that there exists $v \in V$ such that one of the following holds.

– $(L(v), L(e')) \in I$ and $v \rightarrow_c e$. By *N5 in Def. 8, we have that $(v,e) \in \mathcal{E}(L(e'))$ as required.
– $(L(v), L(e)) \in I$ and $e' \rightarrow_c v$. By *N5 in Def. 8, we have that $(e',v) \in \mathcal{E}(L(e))$ as required.

Lemma 4. *Given Petri net N, a g*-LPO $\pi_1 \in \boldsymbol{comp}_g(N)$*, and an* LPO π_2 *such that* $\pi_2 \preceq \pi_1$ *then* $\pi_2 \in \boldsymbol{comp}(N)$*.*

Proof. Given that $\pi_2 \preceq \pi_1$, it follows that both π_1 and π_2 agree on $\rightarrow_c \subseteq V \times E \cup E \times V$ and only disagree in terms of $\rightarrow_i \subseteq E \times E$ and the glue.

It is sufficient to prove that N4, items (b) and (c) in Def. 3 hold for π_2. Consider some $e \in E$. We have the following cases.

- Consider some $v \in V$ and $e \in E$ such that $L(v) \in {}^{\circ}L(e)$. In order to show that $\pi_2 \in \mathbf{comp}(N)$ we have to show that (i) $e \leq v$ or (ii) $e_1, \ldots, e_j$ are all the edges such that $v \rightarrow_c e_k$, we have $\sum_k e_k(v) = i$ and for every k, $e_k < e$. Let e' be the edge such that $e' \rightarrow_c v$ and $f(v)$. By Def. 8 *N5 we have that $(e', v) \in \mathcal{E}(L(e))$. By refinement, we have that either $e \leq e'$, which implies (i) as required; or $v \leq e$ which by $f(v)$ and $\rightarrow_c$ implies (ii). For the case with $\neg f(v)$ we can show by Def. 8 *N5 that we can construct a maximal or infinite order rooted in v with all adjacent elements included in the glue of e and thus it does not allow e to happen after.
- Consider some $v \in V$ and $e \in E$ such that $L(v) \in {}^{\circ}L(e)$. In order to show that $\pi_2 \in \mathbf{comp}(N)$ we have to show that (i) $e \leq v$ or (ii) $e_1, \ldots, e_j$ are all the edges such that $v \rightarrow_c e_k$, we have $\sum_k e_k(v) = i$ and for every k, $e_k < e$. Now, consider every e_k such that $v \rightarrow_c e_k$ and $f(v)$. By Def. 8 *N5 we have that $(v, e_k) \in \mathcal{E}(L(e))$ such that $f(v)$. By refinement, we have that either $e \leq v$; or $e_k \leq e$ as required. The case with $\neg f(v)$ does not apply.
- Consider some $v, v' \in V$ and $e, e' \in E$ such that $v \rightarrow_c e'$, $L(v') \in {}^{\circ}L(e)$ and $v' \leq v$. Let $\neg f(v')$ then by Def. 8 *N5 we have that $(v, e') \in \mathcal{E}(L(e))$. In order to show that $\pi_2 \in \mathbf{comp}(N)$ we have to show that $e \leq v'$ or $e_1, \ldots, e_j$ are all the edges such that $v' \rightarrow_c e_k$, we have $\sum_k e_k(v') = i$ and for every k, $e_k < e$.
  (i) By refinement, we have the option $e \leq v$. We need to prove that (if $e \leq v$ and given $v' \leq v$ then $e \leq v'$). By induction on $v' \leq_k v$, we have that if $k = 0$ then $v' = v$, $(v', e') \in \mathcal{E}(L(e))$ and $e \leq v'$ as required. Assume that if $e \leq v$ and $v' \leq_k v$ then $e \leq v'$ holds and prove it for $k + 1$. We have $v' \leq_k v'' \rightarrow_c e'' \rightarrow_c v$ then by Def. 8 *N5, we have also that $(v'', e''), (e'', v) \in \mathcal{E}(L(e))$ thus, we have that $e \leq e''$ and $e \leq v''$ and by the induction hypothesis $e \leq v'$ as required.
  (ii) By refinement, we have the option $e' \leq e$. we will show that this is not possible for $\neg f(v')$. Let us assume that $e' \leq e$ holds then either $e' \rightarrow_c v'' \not\rightarrow_c$ (i.e., $v''$is maximal) then by Def. 8 *N5 we have that $(e', v'') \in \mathcal{E}(L(e))$, and thus $e \leq v''$ and $e \not\geq v''$, $e \not\geq e'$ as required; there is an infinite order σ starting in e' with every pair of adjacent elements in σ is in $\mathcal{E}(L(e))$ as in Def. 8 *N5, and thus $e \not\geq e'$ as required
- Consider some $v \in V$ and $e, e' \in E$ such that $L(v') \in {}^{\circ}L(e)$, $\neg f(v')$ and $v' \leq e'$. This case is similar to the previous one.
- Consider some $e' \in E$ such that $e \rightarrow_i e'$. By refinement, we have one of the following cases holds.
  - $(e', v) \in \mathcal{E}(L(e))$ for some v. By Def. 8 *N5, we have that $e' \rightarrow_c v$ and $(L(v), L(e)) \in I$ as required.
  - $(v, e) \in \mathcal{E}(L(e'))$ for some v. By Def. 8 *N5, we have that $v \rightarrow_c e$ and $(L(v), L(e')) \in I$ as required.

Theorem 2. *Given a PTI-net N, **comp**$(N) = \{\pi \mid \pi \preceq \pi_g \wedge \pi_g \in \boldsymbol{comp}_g(N)\}$.*

Proof. The proof follows directly from Lemma 3 and Lemma 4.

B Proofs for Section 6 (Separating Choice and Reconfiguration-Forced Interleaving)

Theorem 3. *Given a CTS $\mathcal{T}$ and two different g-*LPO*s $G_1, G_2 \in \boldsymbol{comp}_g(N)$ then one of the following holds:*

1. *For some agent i there exists a history h_i in both G_1 and G_2 such that either h_i is maximal in G_α and not maximal $G_{3-\alpha}$, where $\alpha \in \{1,2\}$;*
2. *For some agent i there exists a history h_i in both G_1 and G_2 such that for the edges e_1 and e_2 such that $h_i \to_{c_1} e_1$ and $h_i \to_{c_2} e_2$ we have $L_1(e_1) \neq L_2(e_2)$;*
3. *or; There is a pair of agents i and i' and histories h_i and $h_{i'}$ in both G_1 and G_2 such that the order between the communications of i and i' is different in G_1 and G_2.*

Proof. We define the *depth* of elements in a partial order as their distance from the minimal element. Formally, the depth of the minimal element is 0 and all the initial states (runs of length 1) have depth of 1. The depth of a non-minimal element o is $\max_{o' \in {}^\bullet o} depth(o') + 1$.

We order the elements in a g-LPO by increasing depth. In addition, elements of the same depth are ordered so that edges appear before vertices and there is some arbitrary order between edges of the same depth and between vertices of the same depth. In this order, every element appears after all the elements that are smaller than it according to $\leq$. Indeed, if $a \to_c b$ or $a \to_i b$, then the depth of b is at least the depth of a plus one. Every element has a finite depth and there is a finite number of elements in every depth. Hence, this order constitutes a linearization of the elements of the g-LPO.

We prove the theorem by induction according to the order mentioned above. We are going to mark elements in the partial order as "equivalent" in both G_1 and G_2.

Consider the minimal element edges in G_1 and G_2 and their post-sets of runs of length 1 (depth 1). By definition, these correspond to the initial states of the different agents. It follows that they are the same. Mark all of them.

Assume that we have marked up to a point in G_1 and G_2 according to the induction order. We build the marking so that the maximal marked elements according to $\leq$ are all nodes. Obviously, all maximal (according to $\leq$) marked elements are incomparable. It follows that we maintain the minimal unmarked element (in induction order) as an edge. Clearly, this is true for the marking of the minimal nodes.

Consider the set of unmarked edges in G_1 and G_2. If both are empty, then G_1 and G_2 are the same. Suppose that the set of unmarked edges in (wlog) G_1 is empty and G_2 is not empty. Consider the sender participating in the communication of the first unmarked edge in G_2. It must be the case that we have found an agent i and a history h_i that is maximal in G_1 and not maximal in G_2. The remaining case is that both G_1 and G_2 have unmarked edges.

Consider the g-LPO G_1. Let e be the minimal unmarked edge in G_1 according to the induction order. Let $h_1, \ldots, h_n$ be ${}^\bullet e$ in G_1 with h_1 being the sender. As all elements of smaller depth than e have been marked, it follows that $h_1, \ldots, h_n$ have been marked and that they appear also in G_2.

Consider a history $h_i \in {}^{\bullet}e$. If h_i is maximal in G_2 we are done. Otherwise, let e_i be the edge such that $h_i \rightarrow_{c_2} e_i$. If $L_1(e) \neq L_2(e_i)$ we are done as h_i does something different in G_1 and G_2. The same holds for every $j \in \{1, \ldots, n\}$. Hence, for every j we have e_j exists and $L(e_j) = L(e)$.

Suppose that G_1 and G_2 are different here. This can only happen if there are at least two agents j and j' for which e_j and $e_{j'}$ are distinct edges labeled by the same communication. In particular, $n \geq 2$ and the agents in histories h_i for $i > 1$ are listening to channel $ch(L(e))$.

However, for e_j and $e_{j'}$ each, there is a unique sender. If h_1 is not sending in G_2 then h_1 does something different in G_1 and G_2 and we are done. Wlog, assume that h_1 is the sender of e_j. Consider the following options.

– Suppose that one of the agents h_i for $i > 1$ is the sender of $e_{j'}$. Then, h_i is a history that receives in G_1 and sends in G_2. Thus, h_i does something different in G_1 and G_2.
– Suppose that there exists an additional agent k and a history h_k such that h_k is the sender for $e_{j'}$. In order *not* to find a difference between G_1 and G_2, it must be the case that h_k is a sender of $e_{j'}$ also in G_1 and the set of agents that participate in e_j and $e_{j'}$ *together* is the same and they have the same roles. That is, every agent that is a receiver in G_1 is a receiver in G_2 and vice versa. However, as we assumed that G_1 and G_2 are different, there are again two options:
    - Either the order between e_j and $e_{j'}$ in G_1 and G_2 is reversed. This matches the difference 3, where the senders are the agents witnessing the difference.
    - Or the order between e_j and $e_{j'}$ is the same in both G_1 and G_2. Then, the matching between senders and receivers in G_1 and G_2 to e_j and $e_{j'}$ is different. Consider a receiver that moved from listening to (wlog) e_j to $e_{j'}$. It follows that this agent participates in an early communication in G_1 (e_j) and a later communication in G_2 ($e_{j'}$). This receiving agent and the sender of e_j see a different order of the communication they participate in (from equal to one before the other).

By induction, unless this process terminates prematurely by finding a difference, it will visit all of G_1 and G_2 and show that they are, in fact, equivalent.

Theorem 4. *Given a Petri net P and two different g-*LPO*s $G_1, G_2 \in \boldsymbol{comp}_g(N)$ then one of the following holds:*

1. *There is a node v_i in both G_1 and G_2 such that the number of tokens not taken from v_i in G_1 and G_2 is different.*
2. *There is a set of p-histories $v_1, \ldots, v_n$ in both G_1 and G_2 that participate in some transition t in G_α but not in $G_{3-\alpha}$, where $\alpha \in \{1, 2\}$.*

Proof. We define the *depth* of a history to be the maximal number of transitions taken by some token in the history. Formally, the depth of $(\emptyset, t_\epsilon)$ is 0. The depth of a p-history (h, p, j) is $depth(h)+1$. For a t-history $e \in E$, let ${}^\bullet e$ be $\{h_1, \ldots, h_n\}$, then the depth of e is $\max_j depth(h_j)$. Notice, that a t-history e could have other edges in its preset.

We order the elements in a g-LPO by increasing depth. In addition, elements of the same depth are ordered so that edges appear before vertices and there is some arbitrary order between edges of the same depth and between vertices of the same depth. Clearly, in this order every element appears after all the elements that are smaller than it according to $\leq$. Indeed, if $a \to_c b$ or $a \to_i b$, then the depth of b is at least the depth of a plus one. As every element has a finite depth and there is a finite number of elements in every depth, it follows that this order is some linearisation of *all* the elements in the g-LPO.

We prove the theorem by induction according to the order mentioned above. We are going to mark nodes and edges that appear in both G_1 and G_2. Nodes are marked by the number of tokens in them that we have not handled yet. When this number is 0 the node is called closed. Otherwise, it is open. Edges are simply marked (or unmarked). For all marked nodes, we "handle" tokens that are participating in the same transitions in G_1 and G_2. Nodes could have tokens that do not participate in transitions. As we "handle" tokens we mark transitions continuing from the node as not forming part of the difference between G_1 and G_2. Once we mark nodes as closed they are also equivalent in G_1 and G_2. As we go through the nodes in G_1 in induction order either we find a difference or, if not, the induction proves that G_1 and G_2 are equivalent in contradiction to the assumption.

Both G_1 and G_2 have the t-history $h_\epsilon = (\emptyset, t_\epsilon)$ as minimal element. Mark it as closed. The p-histories of the form $(h_\epsilon, p, m_0(p))$ such that $m_0(p) > 0$ are marked by $m_0(p)$. Clearly, as both G_1 and G_2 start from the initial marking m_0 both G_1 and G_2 have the same nodes marked and they have the same positive number of tokens.

Assume that we have marked a prefix of G_1 and G_2 such that all closed nodes have all their outgoing transitions marked. Furthermore, the number marking a node is sufficient for all unmarked transitions existing from the node. Clearly, this is true of the marking of the minimal nodes.

Suppose that there are some open nodes. Choose the minimal open node v according to the induction order. If there are no unmarked edges connected to v in both G_1 and G_2 then mark v as closed. If there is no unmarked edge connected to v in G_1 and there is some unmarked edge connected to v in G_2 then we have found a difference as the number of tokens "left" in v in G_1 is larger than in G_2. In this case, we have identified the difference between G_1 and G_2. Similarly for the other way around.

The remaining case is when both in G_1 and G_2 there are unmarked edges connected to v. Let e be the minimal unmarked edge connected to v in G_1. If e is not connected to v in G_2 we are done. Indeed, the preset of e either participate in e in G_1 and not in G_2 or participate in a transition $L(e)$ in different ways in G_1 and G_2.

Otherwise, e is connected to v both in G_1 and G_2. By its construction as a multiset of place histories, e "takes" the same number of tokens from v in G_1 and G_2. As e is unmarked, all the other nodes that e takes tokens from have a sufficient number of unhandled tokens. Again, by e's structure as a pair of a multiset and a transition, e connects to exactly the same nodes in G_1 and G_2 in the same way. Reduce the marking of all predecessors of e by the number of tokens taken by e from them. If some of them are reduced to 0 then they are closed. Mark e as well.

If there are no open nodes, then both G_1 and G_2 are finite and equivalent. Otherwise, continue handling open nodes by induction.

C Additional Materials for Section 3 and Section 4

C.1 Asynchronous automata

A *process* is $P = (\mathsf{Act}, S, s^0, \delta)$, where Act is a finite non-empty alphabet, S is a finite and non-empty set of states, $s^0 \in S$ is an initial state, and $\delta \subseteq S \times \mathsf{Act} \times S$. We also write $\delta : S \times \mathsf{Act} \to 2^S$ when convenient.

A *history* $h = s_0, \ldots, s_n$ is a finite sequence such that $s_0 = s^0$ and for every $0 \leq i < n$ we have $s_{i+1} \in \delta(s_i, a_i)$ for some $a_i \in \mathsf{Act}$. The length of h is $n + 1$, denoted $|h|$. For convenience, if $h_1 = s_0, \ldots, s_n$ and $h_2 = s_0, \ldots, s_n, s_{n+1}$ such that $s_{n+1} \in \delta(s_n, a_n)$ we write $h_2 \in \delta(h_1, a_n)$ or $\delta(h_1, a_n, h_2)$. We define $\mathbf{hist}(P)$ to be the set of histories of P.

A finite asynchronous automaton $\mathcal{A}$ with n processes is $\mathcal{A} = (P_1, \ldots, P_n)$ such that each $P_i = (\mathsf{Act}_i, S_i, s_i^0, \delta_i)$ is a process. Let $\mathsf{Act} = \bigcup_i \mathsf{Act}_i$. A global state of $\mathcal{A}$ is $S = \prod_i S_i$ and the initial state s_0 is $(s_1^0, \ldots, s_n^0)$. The global transition $\Delta \subseteq S \times \mathsf{Act} \times S$ is defined as follows:

$$\Delta = \left\{ ((s_1, \ldots, s_n), a, (s_1', \ldots, s_n')) \;\middle|\; \forall i \;.\; \begin{array}{l} a \in \mathsf{Act}_i \to (s_i, a, s_i') \in \delta_i \text{ and} \\ a \notin \mathsf{Act}_i \to s_i' = s_i \end{array} \right\}$$

Definition 9 (linear computation). *A* linear computation *of $\mathcal{A}$ is a finite or infinite sequence $\pi = s^0, a^0, s^1, a^1, \ldots$ such that $s^0 = s_0$, for every i we have $(s^i, a^i, s^{i+1}) \in \Delta$, and if π is finite it ends in a state.*

We use $\mathbf{comp}_l(\mathcal{A})$ to denote the set of linear computations of $\mathcal{A}$.

C.2 LPO semantics for asynchronous automata

We include the LPO semantics of asynchronous automata. As asynchronous automata do not have reconfigurations of communication we only need the communication relation and do not use the interleaving order relation. This makes the notion of glue not relevant for asynchronous automata.

Consider a finite asynchronous automaton $\mathcal{A}$ with n processes, $\mathcal{A} = (P_1, \ldots, P_n)$, such that each $P_i = (\mathsf{Act}_i, S_i, s_i^0, \delta_i)$ is a process. Let $\mathsf{Act} = \bigcup_i \mathsf{Act}_i$.

Definition 10 (computation). *A* computation *of $\mathcal{A}$ is an* LPO $(O, \rightarrow_c, \Sigma, \Upsilon, L)$*, where $V \subseteq \bigcup_i \boldsymbol{hist}(P_i)$, $\Sigma = V$, $L(h) = h$, and $\Upsilon = \mathsf{Act}$ such that:*

1. *The edge e_ϵ such that $L(e_\epsilon) = a$ for some a is the unique minimal element according to $\leq$. For every i, we have $s_i^0 \in V$ and $e_\epsilon \rightarrow_c s_i^0$.*
2. *If $h \in V$ there is a unique $e \in E$ such that $e \rightarrow_c h$. If $|h| > 1$, there is also a unique $h' \in V$ such that $h \in \delta(h', L(e))$ and $h' \rightarrow_c e$.*
3. *For every $h \in V$ there is at most one $e \in E$ such that $h \rightarrow_c e$.*
4. *For every $e \in E$ there is $I \subseteq [n]$ such that all the following hold:*
   (a) $L_E(e) \in \bigcap_{i \in I} \mathsf{Act}_i \setminus \bigcup_{i \notin I} \mathsf{Act}_i$
   (b) ${}^\bullet e, e^\bullet \in \bigcup_{i \in I} \boldsymbol{hist}(P_i)$
   (c) For every $i \in I$ we have $|{}^\bullet e \cap \boldsymbol{hist}(P_i)| = 1$ and $|e^\bullet \cap \boldsymbol{hist}(P_i)| = 1$

That is, a computation starts from an arbitrary joint edge that leads to the initial states of all processes. For every transition, the set of participating processes is all those having the transition's label in their alphabet. Each participating process has a history that is a predecessor of the transition and a history that is a successor of the transition. The pair of histories that belong to one process satisfy the transition relation of that process.

In particular, the main result that required the introduction of glued partial orders for CTS and Petri nets holds already for LPOs: Every two different LPOs have some process in some (shared) history that does a different action.

D Additional Materials for Section 4 and Section 5

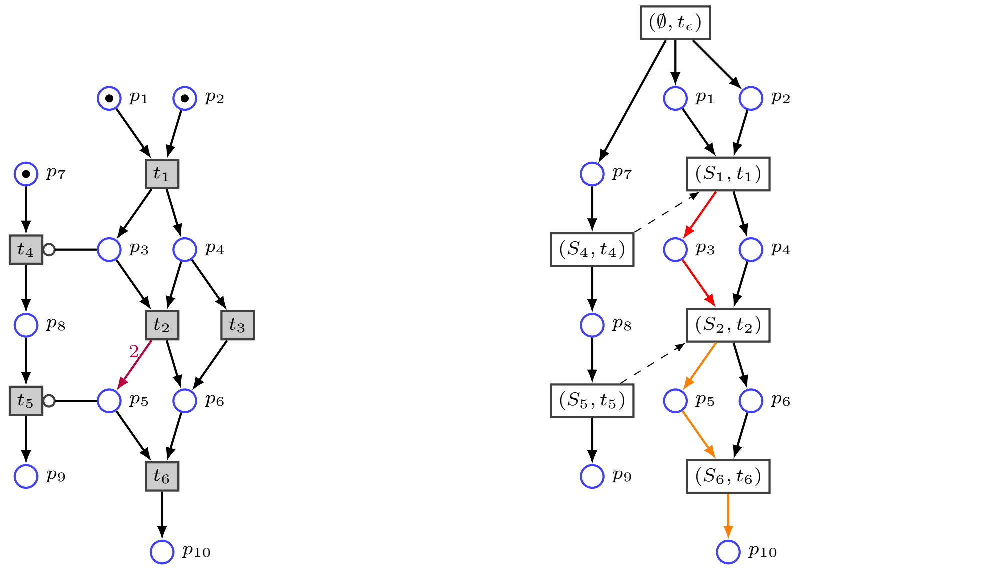
We will use the PTI-net in Fig. 3 (where the multiplicity of unlabelled edges is 1) as a running example to explain our developments and how we handle more intricate cases.

Example 1 (History construction). Consider the PTI-net in Fig. 3(a). As mentioned in Def. 2, every execution is rooted in a special transition t_ϵ with ${t_\epsilon}^\bullet = m_0$. That is why, in Fig. 3(b) both LPOs have a minimal element $(\emptyset, t_\epsilon)$ corresponding to the history of t_ϵ.

Let us focus on the LPO on Fig. 3(b). The history of the place p_i for $i \in \{1, 2, 7\}$ is $h_i = ((\emptyset, t_\epsilon), p_i, {t_\epsilon}^\bullet(p_i)) = (\ldots, p_i, 1)$. Note that ${t_\epsilon}^\bullet(p_i)$ corresponds exactly to the number of tokens in p_i according to the initial marking m_0, namely $m_0(p_i)$. Accordingly, the history of transition t_1 is $(\{(h_1, 1), (h_2, 1)\}, t_1)$.

In the latter case, ${}^\bullet t_1(p_i) = {t_\epsilon}^\bullet(p_i)$ for $i \in \{1, 2\}$. However, the history of t_6 considers the history $h_5 = (h_{t_2}, p_5, 2)$ of place p_5 and the history $h_6 = (h_{t_2}, p_6, 1)$ of place p_6. That is, $h_{t_6} = (\{(h_5, 1), (h_6, 1)\}, t_6)$ and ${}^\bullet t_6(p_5) \leq {t_2}^\bullet(p_5)$. Thus, the other token in p_5 does not belong to the history h_{t_6} (and it is "stuck" in p_5 (or rather h_5) becuase it does not have a matching token in p_6).

*Example 2 (*LPO *construction).* The structure on Fig. 3(b) which represents only one possible LPO, we will consider the construction of such LPO according to Def. 3.



(a) Petri net with Inhibitor arcs (b) One possible LPO (and g-LPO) computations

Fig. 3. Extended PTI-net with nontrivial reconfigurations

The structure on Fig. 3(b) represents an LPO rooted in $(\emptyset, t_\epsilon)$ (according to N1), with $\rightarrow_c$ respecting the structure of the PTI-net (according to N2-N3) and with $\rightarrow_i$ (the dashed arrows) referring to reconfiguration points according to N4 as follows. Notice that we just write the names of places rather than the full p-histories.

LPO : $(S_4, t_4) \rightarrow_i (S_1, t_1)$ and $(S_5, t_5) \rightarrow_i (S_2, t_2)$. That is, the history (S_4, t_4) happens before (S_1, t_1) and the history (S_5, t_5) can only happen before (S_2, t_2). Note that (S_5, t_5) cannot happen after (S_2, t_2) because it does not satisfy N4b which ensures that all tokens are removed from place p_5 before (S_5, t_5) can happen. Note that t_2 in the PTI-net puts two tokens in p_5 and t_6 only consumes one, and thus one token gets stuck in p_5 disabling the execution of t_5 permanently. By restricting attention to interleaving semantics, this LPO gives rise to the following interleavings: $\langle t_4, t_5, t_1, t_2, t_6\rangle$, and $\langle t_4, t_1, t_5, t_2, t_6\rangle$, where t_4 is ordered with respect to the block $\langle t_1, t_2\rangle$ and t_5 is ordered with respect to $\langle t_2, t_6\rangle$.

Example 3 (g-LPO construction). Consider the PTI-net in Fig. 3(a), Def. 8 and the structure on Fig. 3(b) *without the dashed arrows*, which corresponds to a unique g-LPO as follows.

The structure of Fig. 3(b) consists of the LPO from Example 2. Note that the relation $\rightarrow_c$ of the latter LPO is respected according to N1-*N4. Moreover, $(S_4, t_4) \rightarrow_i (S_1, t_1)$ is dropped and is replaced with a glue relation (demonstrated with red colouring) for transition t_4 according to the first two items of $\mathcal{E}(t)$ *N5. Furthermore, $(S_5, t_5) \rightarrow_i (S_2, t_2)$ and the impossibility of (S_5, t_5) happening after

(S_6, t_6) are replaced with a glue relation (demonstrated with orange colouring) for t_5 according to the last three elements of $\mathcal{E}(t)$ *N5.