A stable non-interleaving early operational semantics for the pi-calculus

Hildebrandt, Thomas Troels; Johansen, Christian; Normann, Håkon

Published in:
Journal of Logical and Algebraic Methods in Programming

DOI:
10.1016/j.jlamp.2019.02.006

Publication date:
2019

Document version
Publisher's PDF, also known as Version of record

Citation for published version (APA):
Hildebrandt, T. T., Johansen, C., & Normann, H. (2019). A stable non-interleaving early operational semantics for the pi-calculus. *Journal of Logical and Algebraic Methods in Programming*, *104*, 227-253.
https://doi.org/10.1016/j.jlamp.2019.02.006

Journal of Logical and Algebraic Methods in Programming 104 (2019) 227–253

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

www.elsevier.com/locate/jlamp

A stable non-interleaving early operational semantics for the pi-calculus

Thomas Troels Hildebrandt [a,1], Christian Johansen [b,*,2], Håkon Normann [a,1]

[a] *Department of Computer Science, Copenhagen University, Denmark*
[b] *Institute of Informatics, University of Oslo, Norway*

A R T I C L E   I N F O

Article history:
Received 11 December 2017
Received in revised form 1 August 2018
Accepted 19 February 2019
Available online 21 February 2019

Keywords:
Pi-calculus
Non-interleaving
Early semantics
Asynchronous transition systems
Stability
Causality

A B S T R A C T

We give the first non-interleaving early operational semantics for the pi-calculus which generalises the standard interleaving semantics and unfolds to the stable model of prime event structures. Our starting point is the non-interleaving semantics given for CCS by Mukund and Nielsen, where the so-called *structural* (prefixing or subject) causality and events are defined from a notion of locations derived from the syntactic structure of the process terms. We conservatively extend this semantics with a notion of *extruder histories*, from which we infer the so-called *link* (name or object) causality and events introduced by the dynamic communication topology of the pi-calculus. We prove that the semantics generalises both the standard interleaving early semantics for the pi-calculus and the non-interleaving semantics for CCS. In particular, it gives rise to a labelled asynchronous transition system unfolding to prime event structures.

1. Introduction

The pi-calculus [30] is the seminal model for concurrent mobile processes, representing mobility by the fresh creation and communication of channel names. The standard operational semantics [30,29] adopt an *interleaving* approach to concurrency, that represents concurrent execution of actions as their arbitrary sequential interleaving and employ basic transition systems as semantic models.

However, the ability to distinguish concurrency from interleaving has several practical applications, including dealing with state-space explosion in model-checking [14], supporting action refinement [42], reversibility (e.g. [41,27]), or symmetry [46], besides the gains in expressiveness and modelling accuracy.

To give a non-interleaving semantics one needs to identify the underlying events and their concurrency and causality relationships, and from that define a notion of non-interleaving observations, e.g. in terms of a bisimulation or testing equivalence [38,42] or employ a non-interleaving model (e.g., event structures [34,47], asynchronous transition systems [3, 40] or Petri nets [9]), in which the concurrency can be represented explicitly. The dynamic communication topology of the pi-calculus makes it non-trivial to identify what accounts for causality, and indeed several possible approaches have been proposed. As described in [5], the source of the complexity is that the causal dependencies fall in two categories:

* Corresponding author at: Dept. of Informatics, University of Oslo, P.O. Box 1080 Blindern, 0316 Oslo, Norway.
E-mail addresses: hilde@di.ku.dk (T.T. Hildebrandt), cristi@ifi.uio.no (C. Johansen), normann@di.ku.dk (H. Normann).
[1] The first and last author are supported by the Velux Foundation through the COMPART project (grant 33295).
[2] The second author (previously known as Cristian Prisacariu) was partially supported by the project IoTSec – Security in IoT for Smart Grids, with number 248113/O70 part of the IKTPLUSS program funded by the Norwegian Research Council.

https://doi.org/10.1016/j.jlamp.2019.02.006
2352-2208/

The *structural* (prefixing or subject) dependencies, coming from the static process structure, i.e. action prefixing and parallel composition, and the *link* (name or object) dependencies, which come from the dynamic creation of communication links by scope extrusion of local names.

There has been quite some work on providing non-interleaving semantics for pi-calculus [32,24,9,5,38,12,17,19,18] and process algebras in general (e.g. [8,11]). Among the most recent work, a stable operational semantics for reversible, deterministic and finite pi-calculus processes is provided in [19]. Stability means that every event depends on a unique history of past events, which supports reversibility of computations. More detailed discussions of these related works are provided in Section 5.

A denotational semantics for the pi-calculus is provided in [16,17] as extended event structures. The semantics discards the property of stability to avoid the complexity and increase in number of events arising from achieving unique dependency histories. Both papers consider the late style pi-calculus semantics, where names received from the environment are kept abstract and thus distinct from any previously extruded names. In the early style semantics, names received from the environment are concrete, and thus may be identical to a previously extruded name. Consequently, the choice between late and early style semantics influences the link causality. We found no prior work providing a stable, non-interleaving, *early* style structural operational semantics generalising the standard early operational semantics of the pi-calculus.

Summary of contributions. Our key contribution is to provide the first stable, non-interleaving operational *early* semantics for the pi-calculus that generalises the standard, non-interleaving early operational semantics for the pi-calculus [39]. Our starting point is the work of Mukund and Nielsen [33], which defines a structural operational non-interleaving semantics for Milner's CCS [28] as (labelled) asynchronous transition systems using locations to identify the structural causality, which is the only type of causality in CCS. We generalise the approach of [33] to the pi-calculus by, in addition to the structural locations, employing a notion of *extrusion histories*, recording the location of both name extrusions and name inputs. Together, the locations and extrusion histories allow to identify the underlying events of transitions and both their structural and link causal dependencies.

The notion of extrusion histories was inspired by the recent works of Crafa, Varacca, and Yoshida [16,17]. Their work provides an elegant non-interleaving denotational semantics for the pi-calculus. The elegance is obtained by using a new notion of extended event structures, which allows for non-stable causality, that is, an event does not need to have a unique smallest configuration enabling it. An example of such apparent need for non-stable causality is provided by the process $(\nu n)(\overline{a}\langle n\rangle \,||\, \overline{b}\langle n\rangle \,||\, \underline{n}(x))$, where the input $\underline{n}(x)$ depends on the extrusion of n by one of the concurrent outputs in $\overline{a}\langle n\rangle \,||\, \overline{b}\langle n\rangle$, not both. If one insists on employing a stable non-interleaving model, it seems necessary to have two different (conflicting) events producing the input action $\underline{n}(x)$ representing the two possible dependencies. It is claimed in [16,17] that the technical details become intractable if one insists on employing a stable non-interleaving model. However, the price is that the semantics does not immediately relate to standard, stable models such as asynchronous transition systems and prime event structures.

Overview of paper. In Section 2 we generalise the structural operational early semantics for the pi-calculus with locations for transitions, extrusion histories and link dependencies. In Section 3 we provide some *correctness* results for our semantics, showing that it is a conservative extension of the non-interleaving semantics for CCS and that it conforms to the standard early operational semantics for pi-calculus. We also discuss the differences between early and late semantics. In Section 4 we show that the semantics yields a standard labelled asynchronous transition system, which is known to unfold to labelled prime event structures [47]. We conclude and comment on future work in Section 5.

This paper is an extended version of the LATA2017 conference paper [23]. The present paper works out all the proofs in full detail and adds additional examples and descriptions to explain better our main results. We also add various intermediate helper results and definitions, needed to prove the main results; e.g., the contents from the beginning of Section 3 are needed to prove the main result of Propositions 3.6 and 3.7 which in [23] was only stated as Proposition 8 at the end of Section 2 there. The related works discussions have also been substantially elaborated.

2. Non-interleaving early operational semantics

In this section we give a non-interleaving early operational semantics for the pi-calculus.

We first recall the syntax and standard interleaving early semantics for the pi-calculus with guarded choice.

2.1. Pi-calculus syntax and standard interleaving semantics

Definition 2.1. The set of *pi-calculus processes* **Proc**, ranged over by P, Q, are defined using an infinite set of names $\mathcal{N}$, ranged over by small letters, by the grammar:

$$P ::= \Sigma_{i\in I}\varphi_i.P_i \mid (\nu n)P \mid P \,||\, Q \mid !P \mid \mathbf{0}, \qquad \varphi ::= \overline{a}\langle n\rangle \mid \underline{a}(x)$$

providing constructs for, respectively: guarded non-deterministic choice of output and input prefixes, restriction of name n, parallel composition, replication, and the empty/trivial process. For a process P we denote by $n(P)$ the set of *all names* appearing in P, by $bn(P)$ the *bound* names, i.e. those n that are restricted by (νn) or by the input action $\underline{a}(n)$, and by $fn(P) = n(P)\backslash bn(P)$ the *free* names. E.g. for the process $(\nu n)\underline{a}(x)$ we have $n((\nu n)\underline{a}(x)) = \{n, a, x\}$, $bn((\nu n)\underline{a}(x)) = \{n, x\}$, and $fn((\nu n)\underline{a}(x)) = \{a\}$. We let $P\{x := n\}$ denote the process P where every free occurrence of the name x has been substituted by the name n. We assume all bound names are unique in a process and identify processes up to α-conversion. We make a habit of using $a, b, \dots$ for names that are meant to indicate channels/links for communication, $m, n, \dots$ for names that are being transmitted over channels, and $x, y, z, \dots$ when we intend these names to be substituted. We often omit the indexing set I in the notation for a sum process $\Sigma_{i\in I}\varphi_i.P_i$ and use only $\Sigma\varphi_i.P_i$. When $I = \emptyset$ then we write $\mathbf{0}$ instead of $\Sigma_{i\in\emptyset}\varphi_i.P_i$, and when I is singleton we omit the sum symbol, e.g., $\overline{a}\langle n\rangle.P$. As standard, we often omit the trailing $\mathbf{0}$, e.g. writing $\underline{a}(x).\overline{b}\langle m\rangle$ instead of $\underline{a}(x).\overline{b}\langle m\rangle.\mathbf{0}$.

Next we recall the standard interleaving early semantics for the pi-calculus (see e.g. [37,31]).

Definition 2.2 *(Standard pi-calculus semantic rules).* We denote by $\rightarrow_\pi$ the standard interleaving early semantics given by the following operational rules, ignoring the symmetric versions of the (PAR), (CLOSE) and (COM) rules.

$$\frac{}{\underline{a}(x).P \xrightarrow{\underline{a}(n)}_\pi P\{x := n\}}\ (\text{INP}) \qquad \frac{}{\overline{a}\langle n\rangle.P \xrightarrow{\overline{a}\langle n\rangle}_\pi P}\ (\text{OUT})$$

$$\frac{P \,||\, !P \xrightarrow{\alpha}_\pi P'}{!P \xrightarrow{\alpha}_\pi P'}\ (\text{BANG}) \qquad \frac{\varphi_i.P_i \xrightarrow{\alpha}_\pi P_i'}{\Sigma\varphi_i.P_i \xrightarrow{\alpha}_\pi P_i'}\ (\text{SUM}) \qquad \frac{P \xrightarrow{\alpha}_\pi P' \qquad bn(\alpha)\cap fn(Q) = \emptyset}{P \,||\, Q \xrightarrow{\alpha}_\pi P' \,||\, Q}\ (\text{PAR})$$

$$\frac{P \xrightarrow{\alpha}_\pi P' \qquad b \notin n(\alpha)}{(\nu b)P \xrightarrow{\alpha}_\pi (\nu b)P'}\ (\text{RES}) \qquad \frac{P \xrightarrow{\overline{a}\langle n\rangle}_\pi P' \qquad n \neq a}{(\nu n)P \xrightarrow{(\nu n)\overline{a}\langle n\rangle}_\pi P'}\ (\text{OPEN})$$

$$\frac{P \xrightarrow{(\nu n)\overline{a}\langle n\rangle}_\pi P' \qquad Q \xrightarrow{\underline{a}(n)}_\pi Q'}{P \,||\, Q \xrightarrow{\tau}_\pi (\nu n)(P' \,||\, Q')}\ (\text{CLOSE}) \qquad \frac{P \xrightarrow{\overline{a}\langle n\rangle}_\pi P' \qquad Q \xrightarrow{\underline{a}(n)}_\pi Q'}{P \,||\, Q \xrightarrow{\tau}_\pi P' \,||\, Q'}\ (\text{COM})$$

Note we use the (also standard) notation $(\nu n)\overline{a}\langle n\rangle$ for the bound output (extrusion) of n instead of the notation $\overline{a}(n)$ used in e.g. [37].

Definition 2.3 *(Standard pi generated transition system).* The transition system generated using the standard interleaving, early operational semantics from Definition 2.2 is denoted $TS_\pi = (S, \Lambda, \rightarrow_\pi)$ and defined similarly:

- The set of *states* contains just process terms as in Definition 2.1, i.e., $S \triangleq \{P \mid P \in \mathbf{Proc}\}$;
- The set of labels $\lambda \in \Lambda$ is defined by the grammar $\lambda ::= (\nu n)\overline{m}\langle n\rangle \mid \overline{m}\langle n\rangle \mid \underline{m}(n) \mid \tau$;
- $\rightarrow_\pi \subseteq S \times \Lambda \times S$ are the transitions from Definition 2.2.

For a pi-process P, the transition system reachable from P is denoted $TS_\pi(P)$.

2.2. Early non-interleaving operational semantics

We now give the early non-interleaving operational semantics for pi-processes given by the rules in Fig. 1. The transitions are of the form $(\overline{H}, \underline{H}) \vdash P \xrightarrow[u]{\alpha} (\overline{H}', \underline{H}') \vdash P'$. Compared to the standard semantics above, our semantics employs *location labels* u under the transitions and *extrusion histories* $(\overline{H}, \underline{H})$ to the left of the turnstile. For reasons to be explained below, we have also simplified action labels by not having scope extrusion represented in the labelling. Formally, action labels are defined as follows.

Definition 2.4. The action labels (or simply actions) ranged over by α, β are of the following three kinds:

$\overline{a}\langle n\rangle$	*output*
$\underline{a}(n)$	*input*
τ	*silent*

For an action α, let $n(\alpha)$ denote the set of names appearing in the label, that is, $n(\overline{a}\langle n\rangle) = n(\underline{a}(n)) = \{a, n\}$.

$$\frac{u = [\underline{a}(x).P][P'] \qquad P' = P\{x := n\}}{(\overline{H}, \underline{H}) \vdash \underline{a}(x).P \xrightarrow[u]{\underline{a}(n)} (\overline{H}, \underline{H} \cup \{(n,u)\}) \vdash P'} \;(\textsc{in})$$

$$\frac{u = [\overline{a}\langle n\rangle.P][P]}{(\overline{H}, \underline{H}) \vdash \overline{a}\langle n\rangle.P \xrightarrow[u]{\overline{a}\langle n\rangle} (\overline{H}, \underline{H}) \vdash P} \;(\textsc{out})$$

$$\frac{(\overline{H}, \underline{H}) \vdash P \xrightarrow[u]{\overline{a}\langle n\rangle} (\overline{H}', \underline{H}') \vdash P' \qquad n \neq a}{(\overline{H}, \underline{H}) \vdash (\nu n)P \xrightarrow[u]{\overline{a}\langle n\rangle} (\overline{H}' \cup \{(n,u)\}, \underline{H}') \vdash P'} \;(\textsc{open})$$

$$\frac{(\overline{H}, \underline{H}) \vdash P \xrightarrow[u]{\alpha} (\overline{H}', \underline{H}') \vdash P' \qquad b \notin n(\alpha)}{(\overline{H}, \underline{H}) \vdash (\nu b)P \xrightarrow[u]{\alpha} (\overline{H}', \underline{H}') \vdash (\nu b)P'} \;(\textsc{scope})$$

$$\frac{(\overline{H}, \underline{H}) \vdash P \,||\, !P \xrightarrow[u]{\alpha} (\overline{H}', \underline{H}') \vdash P'}{(\overline{H}, \underline{H}) \vdash !P \xrightarrow[u]{\alpha} (\overline{H}', \underline{H}') \vdash P'} \;(\textsc{rep})$$

$$\frac{(\overline{H}, \underline{H}) \vdash \varphi_i.P_i \xrightarrow[u]{\alpha} (\overline{H}', \underline{H}') \vdash P'}{(\overline{H}, \underline{H}) \vdash \Sigma_{i\in I} : \varphi_i.P_i \xrightarrow[u]{\alpha} (\overline{H}', \underline{H}') \vdash P'} \;(\textsc{sum})$$

$$\frac{\begin{array}{cc} \overline{H}'' = \{(n,u) \mid \exists a.\alpha = \overline{a}\langle n\rangle, n \in \mathsf{dom}([j]\overline{H}), & P_j = P_j' \text{ and } j = 1-i \\ \forall l.l|_{\{0,1\}} \preceq iu : (n,l) \notin \overline{H} \cup \underline{H}\} & \tilde{b} = \mathsf{dom}(\overline{H}_i')\backslash\mathsf{dom}([\check{i}]\overline{H}) \\ ([\check{i}]\overline{H}, [\check{i}]\underline{H}) \vdash P_i \xrightarrow[u]{\alpha} (\overline{H}_i', \underline{H}_i') \vdash P_i' & \tilde{b} \cap fn(P_j) = \emptyset \end{array}}{(\overline{H}, \underline{H}) \vdash P_0 \,||\, P_1 \xrightarrow[iu]{\alpha} \big((\overline{H}\backslash[i]\overline{H}) \cup i\overline{H}_i' \cup i\overline{H}'', (\underline{H}\backslash[i]\underline{H}) \cup i\underline{H}_i'\big) \vdash P_0' \,||\, P_1'} \;(\textsc{par}_i),\ i \in \{0,1\}$$

$$\frac{\begin{array}{cc} \underline{H}'' = \{(n,u_j) \mid & \tilde{b} = \mathsf{dom}(\overline{H}_i')\backslash\mathsf{dom}([\check{i}]\overline{H}) \\ \exists (n,l) \in [i](\overline{H} \cup \underline{H}) : l|_{\{0,1\}} \preceq u_i\} & \tilde{b} \cap fn(P_j) = \emptyset, \quad j = 1-i \\ ([\check{i}]\overline{H}, [\check{i}]\underline{H}) \vdash P_i \xrightarrow[u_i]{\overline{a}\langle n\rangle} (\overline{H}_i', \underline{H}_i') \vdash P_i' & ([\check{j}]\overline{H}, [\check{j}]\underline{H}) \vdash P_j \xrightarrow[u_j]{\underline{a}(n)} (\overline{H}_j', \underline{H}_j') \vdash P_j' \end{array}}{(\overline{H}, \underline{H}) \vdash P_0 \,||\, P_1 \xrightarrow[\langle 0u_0, 1u_1\rangle]{\tau} (\overline{H}, \underline{H} \cup j\underline{H}'') \vdash (\nu\tilde{b})(P_0' \,||\, P_1')} \;(\textsc{com}_i),\ i \in \{0,1\}$$

Fig. 1. Early operational semantics with *action labels* $\alpha ::= \tau \mid \overline{a}\langle n\rangle \mid \underline{a}(n)$, *locations* u (under the arrows) and *extruder histories* $(\overline{H}, \underline{H})$ (to the left of the turnstile). We identify processes up-to α-equivalence, assume unique bound names, and require for all rules, if $(\overline{H}, \underline{H}) \vdash P \xrightarrow[u]{\alpha} (\overline{H}', \underline{H}') \vdash P'$ is the conclusion, then $\mathsf{dom}(\overline{H} \cup \underline{H}) \cap bn(P) = \emptyset$. For rules $(\textsc{com}_i)$ where $i \in \{0,1\}$, we allow writing $(\nu\emptyset)P$ and $(\nu\{n\})P$ for P and $(\nu n)P$ respectively. The blue text shows what is added to the standard semantics. (For interpretation of the colours in the figure and text, the reader is referred to the web version of this article.)

Recalling from [33], the set of prefix locations, i.e. locations of action prefixes inside process terms, is defined as follows.

Definition 2.5. Let $\mathcal{L} = \{0,1\}^* \times \textbf{Proc} \times \textbf{Proc}$ be the set of *prefix locations* and write $s[P][P']$ for triples $(s, P, P') \in \mathcal{L}$. For $l = s[P][P']$ let $l|_{\{0,1\}} = s$ and let $\preceq$ be the (reflexive) prefix order on $\{0,1\}^*$. We use $\epsilon \in \{0,1\}$ as the empty string, which we often omit and only write $[P][P']$ instead of $\epsilon[P][P']$.

Proposition 2.6 *(Location labels). The location label u of a transition $(\overline{H}, \underline{H}) \vdash P \xrightarrow[u]{\alpha} (\overline{H}', \underline{H}') \vdash P'$ derived by the rules in Fig. 1 has the form:*

1. *$u = s[P][P'] \in \mathcal{L}$, if α is an input or output action,*
2. *$u = s\langle 0s_0[P_0][P_0'], 1s_1[P_1][P_1']\rangle$, for $si s_i[P_i][P_i'] \in \mathcal{L}$ and $i \in \{0,1\}$, if $\alpha = \tau$.*

Proof. By induction on the length of the derivation tree of the transition $(\overline{H}, \underline{H}) \vdash P \xrightarrow[u]{\alpha} (\overline{H}', \underline{H}') \vdash P'$. $\square$

$$\begin{tikzcd}
& \overline{a}\langle a\rangle.\overline{b}\langle b\rangle + \overline{b}\langle b\rangle.\overline{a}\langle a\rangle \arrow[dl, "\overline{a}\langle a\rangle"', "{[\overline{a}\langle a\rangle.\overline{b}\langle b\rangle][\overline{b}\langle b\rangle]}"] \arrow[dr, "\overline{b}\langle b\rangle", "{[\overline{b}\langle b\rangle.\overline{a}\langle a\rangle][\overline{a}\langle a\rangle]}"'] & \\
\overline{b}\langle b\rangle \arrow[dr, "\overline{b}\langle b\rangle", "{[\overline{b}\langle b\rangle][\mathbf{0}]}"'] & & \overline{a}\langle a\rangle \arrow[dl, "\overline{a}\langle a\rangle"', "{[\overline{a}\langle a\rangle][\mathbf{0}]}"] \\
& \mathbf{0} &
\end{tikzcd}$$

(a) Interleaving.

$$\begin{tikzcd}
& \overline{a}\langle a\rangle \,||\, \overline{b}\langle b\rangle \arrow[dl, "\overline{a}\langle a\rangle"', "u_1"] \arrow[dr, "\overline{b}\langle b\rangle", "u_2"'] & \\
\mathbf{0} \,||\, \overline{b}\langle b\rangle \arrow[dr, "\overline{b}\langle b\rangle", "u_2"'] & & \overline{a}\langle a\rangle \,||\, \mathbf{0} \arrow[dl, "\overline{a}\langle a\rangle"', "u_1"] \\
& \mathbf{0} \,||\, \mathbf{0} &
\end{tikzcd}$$

for $u_1 = 0[\overline{a}\langle a\rangle][\mathbf{0}]$ and $u_2 = 1[\overline{b}\langle b\rangle][\mathbf{0}]$

(b) Concurrency.

Fig. 2. Non-interleaving vs. interleaving diamond distinguished through different location labels.

In words, a prefix location $s[P][P']$ provides a path $s \in \{0,1\}^*$ to an input/output prefixed subterm P through the abstract syntax tree, with 0 and 1 referring to the left respectively right component of a parallel composition, and P' being the residual sub-term after the transition. A location label of the form $s\langle 0s_0[P_0][P'_0], 1s_1[P_1][P'_1]\rangle$ provides two prefix locations, $ss_i[P_i][P'_i] \in \mathcal{L}$ for $i \in \{0,1\}$, identifying the location of the output and the input prefix interacting in a communication action.

The location labels identify the location of the prefixes in the term contributing to a transition and allowing to infer structural (CCS-like) events and causalities following the approach in [33]. The extrusion histories keep track of the location of extrusion and reception of names. As we will prove later, if one ignores extrusion histories and location labels, the semantics is almost the standard early semantics. The only difference is in the labelling of actions in the open rule, where we do not add the extruded name to the label as in the standard semantics. There are two reasons for doing this, as will become clear later: Firstly, since the extruder histories capture the extrusion, there is no need for recording it in the label. Secondly, if the extruded name is recorded in the label, the same event can occur in the transition system two places with different labels. That is, the labelling of events will not be unique and we could not give a direct semantics in terms of standard labelled asynchronous transition systems.[3]

The extrusion histories are defined formally as follows.

Definition 2.7 *(Extrusion histories).* A *history* $H \subseteq \mathcal{N} \times \mathcal{L}$ is a relation between names and prefix locations. An *extrusion history* $(\overline{H}, \underline{H})$ is a pair of histories, referred to as the *output history* and *input history* respectively. The output history $\overline{H}$ records for each extruded name the locations of the prefixes that extruded it. The input history $\underline{H}$ records for each name n received by an input prefix the locations of the input prefixes that received the name. Let $\mathsf{dom}(H) = \{n \mid (n,u) \in H\}$, i.e. the set of names recorded in the history.

To manipulate histories we define the following operations.

Definition 2.8 *(Operations on extrusion histories).* For a history $H \subseteq \mathcal{N} \times \mathcal{L}$ and $i \in \{0,1\}$, let

- $iH = \{(n, iu) | (n,u) \in H\}$,
- $[\check{i}]H = \{(n,u) \mid (n,iu) \in H\}$,
- $[i]H = \{(n,iu) \mid (n,iu) \in H\}$,
- $[\check{\epsilon}]H = H$.

We may apply the above operations on histories several times and abbreviate them by a string, as in the example $[\check{0s}]H = [\check{0}][\check{s}]H$ with s a possibly empty location string.

We now describe the use of locations and histories to capture structural and link dependencies in the semantics.

2.3. Structural dependencies

We begin with an example showing how locations are used to record the difference between the choice between interleavings of two actions and the parallel execution of two structurally independent actions.

Example 2.9 *(Non-interleaving vs. interleaving).* Fig. 2 shows the transition semantics of the Pi process $\overline{a}\langle a\rangle.\overline{b}\langle b\rangle + \overline{b}\langle b\rangle.\overline{a}\langle a\rangle$ ("$\overline{a}\langle a\rangle$ then $\overline{b}\langle b\rangle$ or $\overline{b}\langle b\rangle$ then $\overline{a}\langle a\rangle$") to the left and the Pi process $\overline{a}\langle a\rangle \,||\, \overline{b}\langle b\rangle$ ("$\overline{a}\langle a\rangle$ and in parallel $\overline{b}\langle b\rangle$") to the right,

[3] In this case one would e.g. need to use an equivalence over the action labels which would have complicated the technical development in several places.

enriched with locations under the transition arrows. If one can only observe the transitions and action labels then it is not possible to tell the difference. However, the location labels under the transitions make it possible to see that in the process to the left, all transitions are due to different prefixes located at the top location, while in the process to the right, the two a-transitions are due to the same prefix at the location u_1 and the two b-transitions are due to the same prefix at the location u_2. Finally, the 0 and the 1 in the paths of location u_1 and u_2 show that the two locations are in two different components of a parallel composition. This is the basis for defining events and the structural independence relation on events below.

From the locations we define our first notion of structural events and independence, which correspond to the events respectively independence relation defined for CCS in [33]. We will later show how to take into account the additional link causal relationships of the pi-calculus.

Definition 2.10 *(Structural events).* Let $Ev = \{(\alpha, u) \mid \exists P, P', \overline{H}, \underline{H}, \overline{H}', \underline{H}' : (\overline{H}, \underline{H}) \vdash P \xrightarrow[u]{\alpha} (\overline{H}', \underline{H}') \vdash P'\}$ be the *structural events*. For $e = (\alpha, u) \in Ev$ define $Loc(e) \subseteq \{0,1\}^*$, the *locations* where e occurs, by

$$Loc(e) = \begin{cases} \{s\} & \text{if } u = s[P][P'] \\ \{ss_0, ss_1\} & \text{if } u = s\langle s_0[P_0][P_0'], s_1[P_1][P_1']\rangle. \end{cases}$$

Note that to keep locations of action prefixes fixed, we cannot assume the usual structural congruence making parallel composition commutative. Therefore we must use two rules (PAR_i), $i \in \{0,1\}$, for parallel composition and two rules (COM_i), $i \in \{0,1\}$, for communication.

Example 2.11 *(Associativity and commutativity break location labels).* Often in presentations of pi-calculus one considers structural rules like for the parallel composition being commutative and associative. We cannot have such structural rules because of how we create the location labels of the events and how we define the independence relation on events in Definition 2.12. For the commutativity of parallel composition we can easily see how our labels do not work any more, in the example process $P = \overline{a}\langle a\rangle \,||\, \overline{b}\langle b\rangle$. Here we can get the event $e = (\overline{a}\langle a\rangle, 0[\overline{a}\langle a\rangle][\mathbf{0}])$ for the transition on channel a. If we had a structural rule saying that parallel composition can commute, i.e., that the two processes $P = \overline{a}\langle a\rangle \,||\, \overline{b}\langle b\rangle = \overline{b}\langle b\rangle \,||\, \overline{a}\langle a\rangle = P'$ are equivalent for the semantic rules, we would see $P' = \overline{b}\langle b\rangle \,||\, \overline{a}\langle a\rangle$ executing the same event for the same channel a but which has a different location label $e = (\overline{a}\langle a\rangle, 1[\overline{a}\langle a\rangle][\mathbf{0}])$. This breaks our reasoning about which components execute which events. For the associativity of parallel consider the process $P' = (\overline{a}\langle a\rangle \,||\, \overline{b}\langle b\rangle) \,||\, \overline{c}\langle c\rangle$ where the event for b would be $(\overline{b}\langle b\rangle, 01[\overline{b}\langle b\rangle][\mathbf{0}])$. If we allow the parallel to be associative we could write $P' = \overline{a}\langle a\rangle \,||\, (\overline{b}\langle b\rangle \,||\, \overline{c}\langle c\rangle)$ where the event for b would be $(\overline{b}\langle b\rangle, 10[\overline{b}\langle b\rangle][\mathbf{0}])$. These two events again have completely different location labels, though they should be seen as coming from the same component.

Definition 2.12 *(Structural independence).* Define an *independence relation on locations* $I_l \subseteq \{0,1\}^* \times \{0,1\}^*$ by

$$(s_0, s_1) \in I_l \quad \text{iff} \quad (s_0 = s0s_0' \wedge s_1 = s1s_1') \vee (s_0 = s1s_0' \wedge s_1 = s0s_1'),$$

where $s, s_0, s_1, s_0', s_1' \in \{0,1\}^*$. Define the *structural independence* relation on events $I_s \subseteq Ev \times Ev$ by:

$$(e, e') \in I_s \quad \text{iff} \quad \forall s \in Loc(e), \forall s' \in Loc(e') : (s, s') \in I_l.$$

Note how the definition of independence relation correctly identifies the two events $(\overline{a}\langle a\rangle, u_1)$ and $(\overline{b}\langle b\rangle, u_2)$ from Fig. 2b as being independent. Thus, the transitions form a correct independence diamond in the sense of labelled asynchronous transition systems (see definition later). Moreover, the four events from Fig. 2a are not independent as they all have the same location path identified by the empty string. Working in this way with location labels also handles correctly recursion as explained nicely in [33, Fig. 2].

2.4. Link dependencies

The structural independence relation I_s as defined on structural events above misses the link dependencies resulting from extrusion of names. The following example shows a simple link dependency and how it is captured by the extrusion histories and refining the events.

Example 2.13 *(Simple link dependency).* Consider the process

$$(\nu n)(\overline{a}\langle n\rangle \,||\, \underline{n}(x)),$$

$$\begin{tikzcd}
 & (\{(n,u)\},\emptyset) \vdash \mathbf{0} \,||\, \underline{n}(x) \arrow[dr, "\underline{n}(m)", "u'"'] & \\
(\emptyset,\emptyset) \vdash (\nu n)(\overline{a}\langle n\rangle \,||\, \underline{n}(x)) \arrow[ur, "\overline{a}\langle n\rangle", "u"'] & & (\{(n,u)\},\{(m,u')\}) \vdash \mathbf{0} \,||\, \mathbf{0}
\end{tikzcd}$$

Fig. 3. Simple link dependency described in Example 2.13, where $u = 0[\overline{a}\langle n\rangle][\mathbf{0}]$ and $u' = 1[\underline{n}(x)][\mathbf{0}]$.

with its behaviour pictured in Fig. 3. According to I_s the two events $e = (\overline{a}\langle n\rangle, u)$ and $e_m = (\underline{n}(m), u')$, for $u = 0[\overline{a}\langle n\rangle][\mathbf{0}]$ and $u' = 1[\underline{n}(x)][\mathbf{0}]$, are structurally independent.[4] The semantics, however, does not allow the input event to happen until after the name n has been extruded by the output event. As also observed in e.g. [17,20], there is a link dependency between the extruding output and the input. The extrusion histories allow for capturing the link causalities of the pi-calculus. Only the output history is needed to capture the link causality in this simple example. Initially, the input and output histories are empty, i.e. $\underline{H} = \overline{H} = \emptyset$. The (OPEN) rule records the extrusion of the name n in the output history, resulting in the extrusion history $(\{(n,u)\},\emptyset)$. For now we can ignore the complicated pre-conditions of the (PAR) rule, since when $\underline{H} = \overline{H} = \emptyset$ it follows that $[\check{i}]\overline{H} = H'' = [i]\overline{H} = \emptyset$. For the following input transition, the (IN) records the location of the prefix receiving the name, which we will need to handle the more complex situations of parallel extrusion possible in the late semantics to be explained below. To capture the link dependency from the events (and ensure proper splitting of events, to be explained later) we enrich in Definition 2.18 below the location under the transitions to include also the location u where the name n was extruded (as recorded in the output history): $(\{(n,u)\},\emptyset) \vdash \mathbf{0} \,||\, \underline{n}(x) \xrightarrow[u'\{(n,u)\}]{\underline{n}(m)} (\{(n,u)\},\{(m,u')\}) \vdash \mathbf{0} \,||\, \mathbf{0}$. This extra information is used to split events and refine the structural independence relation to a causal independence relation, as defined formally in Definition 2.20. The causal independence relation does not relate the two events $e = (\overline{a}\langle n\rangle, u)$ and $e'_m = (\underline{n}(m), u'\{(n,u)\})$, since the latter can be seen to be link dependent on the first.

Extrusion of names in the above example is immediate in the (OPEN) rule. However, as shown by the next example, other events in parallel, under the same scope, may also be extruders, and we capture this in the (PAR) rule. We will refer to the immediate extruders inferred by the (OPEN) as *initial extruders* and to the latter as *possible extruders*.

Example 2.14 *(Parallel extruders).* Consider the process $(\nu n)(\overline{a}\langle n\rangle \,||\, (\overline{b}\langle n\rangle \,||\, \underline{n}(x)))$, which has two independent *parallel extruders* $\overline{a}\langle n\rangle$ and $\overline{b}\langle n\rangle$. According to Definition 2.10 we have the three structural events $e_a = (\overline{a}\langle n\rangle, 0[\overline{a}\langle n\rangle][\mathbf{0}])$, $e_b = (\overline{b}\langle n\rangle, 10[\overline{b}\langle n\rangle][\mathbf{0}])$, and $e_m = (\underline{n}(m), 11[\underline{n}(x)][\mathbf{0}])$.[5] The event e_m for the input action is structurally independent of both output events, but it cannot happen before at least one has happened.

If we only captured extrusions in the (OPEN) rule, the output history after the two output actions would be $\{(n, 0[\overline{a}\langle n\rangle][\mathbf{0}])\}$ if the output transition on the name a is taken first, but $\{(n, 10[\overline{b}\langle n\rangle][\mathbf{0}])\}$ if the output transition on the name b is taken first. This means that the resulting state would depend on the order, and thus contradict that the two events are independent. The set H'' in the premise of the (PAR) rule records the parallel extruders. Let's assume the output transition on the name a is taken first, giving the transition $(\emptyset,\emptyset) \vdash (\nu n)(\overline{a}\langle n\rangle \,||\, (\overline{b}\langle n\rangle \,||\, \underline{n}(x))) \xrightarrow[u]{\overline{a}\langle n\rangle} (\{(n,u)\},\emptyset) \vdash \mathbf{0} \,||\, (\overline{b}\langle n\rangle \,||\, \underline{n}(x))$ for $u = 0[\overline{a}\langle n\rangle][\mathbf{0}]$. When inferring the following output transition on the name b we get

$$(\{(n,u)\},\emptyset) \vdash \mathbf{0} \,||\, (\overline{b}\langle n\rangle \,||\, \underline{n}(x)) \xrightarrow[u']{\overline{b}\langle n\rangle} (\{(n,u),(n,u')\},\emptyset) \vdash \mathbf{0} \,||\, (\mathbf{0} \,||\, \underline{n}(x))$$

for $u' = 10[\overline{b}\langle n\rangle][\mathbf{0}]$ using the (PAR$_1$) rule. To see this, note that we have in the conclusion of the rule $\overline{H} = \{(n,u)\}$, $\underline{H} = \emptyset$, $\alpha = \overline{b}\langle n\rangle$ and $iu = 10[\overline{b}\langle n\rangle][\mathbf{0}]$. The transition is then inferred from the following premises $H'' = \{(n, 0[\overline{b}\langle n\rangle][\mathbf{0}])\}$, because $\exists a.\alpha = \overline{a}\langle n\rangle$ and $n \in \mathrm{dom}([0]\overline{H})$ and there is no $(n,l) \in \overline{H} \cup \underline{H} = \{(n,u)\}$ such that $l|_{\{0,1\}} \prec 10[\overline{b}\langle n\rangle][\mathbf{0}]$. The condition that there is no $(n,l) \in \overline{H} = \{(n,u)\}$ such that $l|_{\{0,1\}} \prec 10[\overline{b}\langle n\rangle][\mathbf{0}]$ ensures e.g. that the output transition on the name c in the following processes is *not* considered a parallel extruder of the output transition on a: $(\nu n)(\overline{a}\langle n\rangle \,||\, (\overline{b}\langle n\rangle.\overline{c}\langle n\rangle \,||\, \underline{n}(x)))$ and $(\nu n)(\overline{a}\langle n\rangle \,||\, \overline{b}\langle n\rangle.(\overline{c}\langle n\rangle \,||\, \underline{n}(x)))$. The condition that there is no $(n,l) \in \underline{H} = \{(n,u)\}$ such that $l|_{\{0,1\}} \prec 10[\overline{b}\langle n\rangle][\mathbf{0}]$ handles a case where the name has been received before extruded, which is also illustrated in Example 2.16. In the context of the present example, it means that the output transition on the name c in the following processes: $(\nu n)(\overline{a}\langle n\rangle \,||\, (\underline{b}(m).\overline{c}\langle n\rangle \,||\, \underline{n}(x)))$ is *not* considered a parallel extruder of the output transition on a, if the name received on b is the previously extruded name n.

Now, we want that the following input on n is disjunctively dependent on the two extruders, but we cannot have that in a stable semantics. The solution is to split the input event e_m in two events e_{ma} and e_{mb}, one for each parallel extruder, i.e. we get from Definition 2.18 the two input transitions

$$(\{(n,u),(n,u')\},\emptyset) \vdash \mathbf{0} \,||\, (\mathbf{0} \,||\, \underline{n}(x)) \xrightarrow[u''\{(n,u)\}]{\underline{n}(m)} (\{(n,u),(n,u')\},\{(m,u'')\}) \vdash \mathbf{0} \,||\, (\mathbf{0} \,||\, \mathbf{0})$$

[4] Strictly speaking there are infinitely many input events on n, one for each possible substitution $x := m$.

[5] As in the previous example, we in fact have infinitely many input events on n, one for each possible substitution $x := m$.

and

$$(\{(n,u),(n,u')\},\emptyset) \vdash \mathbf{0} \,||\, (\mathbf{0} \,||\, \underline{n}(x)) \xrightarrow[u''\{(n,u')\}]{\underline{n}(m)} (\{(n,u),(n,u')\},\{(m,u'')\}) \vdash \mathbf{0} \,||\, (\mathbf{0} \,||\, \mathbf{0})$$

for $u'' = 11[\underline{n}(x)][\mathbf{0}]$. The event $e_{ma} = (\underline{n}(m), u''\{(n,u)\})$ is dependent on the extruder on a and independent of the extruder on b, while the event $e_{mb} = (\underline{n}(m), u''\{(n,u')\})$ is dependent on the extruder on b and independent of the extruder on a. Finally, the two input events will not be independent of each other, since they happen at the same location path.

As shown in Example 2.14 above, instead of a single event e_m for the input event, we use the extrusion histories to split the event e_m in two conflicting events, e_{ma} and e_{mb}, one causally dependent on output event e_a and one dependent on output event e_b. Crafa et al. in [17, Sec. 6] deem the approach of splitting events intractable and instead follow a different approach by defining a specially tailored, non-stable event structure model incorporating a global set of extruded names. Our approach, however, works with the standard stable model of asynchronous transition systems (which unfolds to prime event structures [47, Ch. 10]) since we split events involved in disjunctive causalities using the insights from [17].

The next example illustrates that in an early semantics both names in an input action $\underline{m}(n)$ may have been previously extruded. In other words, the early semantics allow a form of *conjunctive causality* that cannot happen in the late semantics.

Example 2.15 *(Link dependency on two names in one action in early semantics).* Consider the process $P = (\nu n)(\nu m)(\overline{a}\langle n\rangle \,||\, (\overline{b}\langle m\rangle \,||\, \underline{m}(x)))$ from which there are two extruding outputs on channels a respectively b of the names n respectively m. We now may have an input action with label $\underline{m}(n)$ that depends on both extruders. We also have input actions with labels $\underline{m}(n')$ for $n \neq n'$ that only depend on the extruder at location $10[\overline{b}\langle m\rangle][\mathbf{0}]$. In the late semantics, we only have the latter situation, since the name being input can not be a previously extruded name.

We illustrate the use of the input history in Example 2.16.

Example 2.16. Consider the process $P = (\nu n)(\overline{a}\langle n\rangle \,||\, \underline{b}(x).\overline{c}\langle n\rangle)$. Starting with empty histories, we have the two transitions

1. $(\emptyset,\emptyset) \vdash P \xrightarrow[0[\overline{a}\langle n\rangle][\mathbf{0}]]{\overline{a}\langle n\rangle} (\{(n, 0[\overline{a}\langle n\rangle][\mathbf{0}])\},\emptyset) \vdash P_1$, for $P_1 = \mathbf{0} \,||\, \underline{b}(x).\overline{c}\langle n\rangle$
2. $(\emptyset,\emptyset) \vdash P \xrightarrow[1[\underline{b}(x).\overline{c}\langle n\rangle][\overline{c}\langle n\rangle]]{\underline{b}(m)} (\emptyset, \{(m, 1[\underline{b}(x).\overline{c}\langle n\rangle][\overline{c}\langle n\rangle])\}) \vdash (\nu n)(\overline{a}\langle n\rangle \,||\, \overline{c}\langle n\rangle)$, with $m \neq n$.

After the first transition we may receive either n or an arbitrary name $m \neq n$:

$$(\{(n, 0[\overline{a}\langle n\rangle][\mathbf{0}])\},\emptyset) \vdash P_1 \xrightarrow[1[\underline{b}(x).\overline{c}\langle n\rangle][\overline{c}\langle n\rangle]]{\underline{b}(n)} (\{(n, 0[\overline{a}\langle n\rangle][\mathbf{0}])\}, \{(n, 1[\underline{b}(x).\overline{c}\langle n\rangle][\overline{c}\langle n\rangle])\}) \vdash (\mathbf{0} \,||\, \overline{c}\langle n\rangle).$$

$$(\{(n, 0[\overline{a}\langle n\rangle][\mathbf{0}])\},\emptyset) \vdash P_1 \xrightarrow[1[\underline{b}(x).\overline{c}\langle n\rangle][\overline{c}\langle n\rangle]]{\underline{b}(m)} (\{(n, 0[\overline{a}\langle n\rangle][\mathbf{0}])\}, \{(m, 1[\underline{b}(x).\overline{c}\langle n\rangle][\overline{c}\langle n\rangle])\}) \vdash (\mathbf{0} \,||\, \overline{c}\langle n\rangle).$$

In the first case, a subsequent output of n on channel c will not be an extrusion, since it happens after the input of n from the environment. In the second case it will, since this output is independent of the extrusion in transition 1. The input history thus allows the (PAR$_i$) rules to distinguish between outputs that knew of n before the scope of n was opened and outputs that learned of n by receiving it after it was extruded.

Finally we illustrate the use of the input history in the (COM) rule in Example 2.17 below, based on a process also appearing in [17].

Example 2.17. Consider the process $P = (\nu n)\big((\overline{a}\langle n\rangle \,||\, \overline{b}\langle n\rangle) \,||\, (\underline{b}(x).\overline{c}\langle x\rangle \,||\, \underline{n}(y))\big)$ given in [17].

The first point to note is that the name n can be extruded by three actions: 1) The $\overline{a}\langle n\rangle$ action at location $00[\overline{a}\langle n\rangle][\mathbf{0}]$, 2) the $\overline{b}\langle n\rangle$ action at location $01[\overline{b}\langle n\rangle][\mathbf{0}]$ or 3) the $\overline{c}\langle n\rangle$ action at location $10[\overline{c}\langle n\rangle][\mathbf{0}]$ if we first have an internal communication on the channel b. This means that we have three different events for the input action $\underline{n}(y)$. The next point to note is that either $\overline{a}\langle n\rangle$ and $\overline{b}\langle n\rangle$ are parallel extruders of n *or*, $\overline{a}\langle n\rangle$ and $\overline{c}\langle n\rangle$ (after the internal communication on channel b).

To illustrate the use of the input history in the (COM) rule, we consider the transition sequence beginning with the extrusion of n by the action $\overline{a}\langle n\rangle$ followed by the internal communication on the channel b. The first transition updates the output history with the location of the extruding action as follows:

$(\emptyset,\emptyset) \vdash P \xrightarrow[00[\overline{a}\langle n\rangle][\mathbf{0}]]{\overline{a}\langle n\rangle} (\{(n, 00[\overline{a}\langle n\rangle][\mathbf{0}])\},\emptyset) \vdash P_1$, for $P_1 = (\mathbf{0} \,||\, \overline{b}\langle n\rangle) \,||\, (\underline{b}(x).\overline{c}\langle x\rangle \,||\, \underline{n}(y))$

The second transition, using the (COM) rule, does not update the histories:

$(\{(n, 00[\overline{a}\langle n\rangle][\mathbf{0}])\},\emptyset) \vdash P_1 \xrightarrow[\langle 01[\overline{b}\langle n\rangle][\mathbf{0}],10[\underline{b}(n).\overline{c}\langle x\rangle][\overline{c}\langle n\rangle]\rangle]{\tau} (\{(n, 00[\overline{a}\langle n\rangle][\mathbf{0}])\},\emptyset) \vdash (\mathbf{0} \,||\, \mathbf{0}) \,||\, (\overline{c}\langle x\rangle \,||\, \underline{n}(y)).$

Looking at the premises of the rule, we get that the set $H'' = \emptyset$, since n has not been extruded or received at a prefix of the location output action $\overline{b}\langle n\rangle$. This means that the action $\overline{c}\langle n\rangle$ will be detected as a parallel extruder by the definition of H'' in the (PAR) rule.

Consider now the example process $P' = (\nu n)\big((\overline{a}\langle n\rangle \,||\, \overline{d}\langle n\rangle.\overline{b}\langle n\rangle) \,||\, (\underline{b}(x).\overline{c}\langle x\rangle \,||\, \underline{n}(y))\big)$ and assume again we have executed the $\overline{a}\langle n\rangle$ as the first transition but then also executed the $\overline{d}\langle n\rangle$ transition, resulting in the following process $H \vdash P''$ for $P'' = (\mathbf{0} \,||\, \mathbf{0}.\overline{b}\langle n\rangle) \,||\, (\underline{b}(x).\overline{c}\langle x\rangle \,||\, \underline{n}(y))$ and extrusion history

$$H = (\{(n, 00[\overline{a}\langle n\rangle][\mathbf{0}]), (n, 01[\overline{d}\langle n\rangle.\overline{b}\langle n\rangle][\overline{b}\langle n\rangle])\}, \emptyset),$$

having two parallel extruders recorded in the output history.

Now the internal communication using the (COM) rule will update the input history:
$H \vdash P'' \xrightarrow[\langle 01[\overline{b}\langle n\rangle][\mathbf{0}],10[\underline{b}(n).\overline{c}\langle x\rangle][\overline{c}\langle n\rangle]\rangle]{\tau} H' \vdash (\mathbf{0} \,||\, \mathbf{0}) \,||\, (\overline{c}\langle x\rangle \,||\, \underline{n}(y))$ for

$$H' = \big(\{(n, 00[\overline{a}\langle n\rangle][\mathbf{0}]), (n, 01[\overline{d}\langle n\rangle.\overline{b}\langle n\rangle][\overline{b}\langle n\rangle])\}, \{(n, 10[\underline{b}(n).\overline{c}\langle x\rangle][\overline{c}\langle n\rangle])\}\big),$$

since $H'' = \{(n, 10[\underline{b}(n).\overline{c}\langle x\rangle][\overline{c}\langle n\rangle])\}$ in the premise of the (COM) rule.

The fact that the reception of the name n is recorded in the input history now *prevents* the action $\overline{c}\langle n\rangle$ being treated as a parallel extruder by the definition of H'' in the (PAR) rule. Finally, note that this example holds for both the early and late semantics.

Based on the examples above, we refine our transitions and events to also capture link dependencies by enriching the transitions with a set D, which we call deterministic sub-history, recording the link dependencies for each non-output name in α, i.e. the past extruding events a name depends on, if it was extruded in the past.

Definition 2.18 *(Causal semantics).* Define the *causal* early semantics as the following transitions: $(\overline{H}, \underline{H}) \vdash P \xrightarrow[u,D]{\alpha} (\overline{H}', \underline{H}') \vdash P'$ if $(\overline{H}, \underline{H}) \vdash P \xrightarrow[u]{\alpha} (\overline{H}', \underline{H}') \vdash P'$ and

1. $D \subseteq \overline{H}$,
2. $(n, l), (n, l') \in D$ implies $l = l'$,
3. $\mathrm{dom}(D) = \mathrm{dom}(\overline{H}) \cap no(\alpha)$,

where $no(\alpha)$ is the *non-output names* of α, defined by $no(\overline{n}\langle m\rangle) = \{n\}\backslash\{m\}$, $no(\underline{n}(m)) = \{n, m\}$ and $no(\tau) = \emptyset$.

The set D contains only one entry per name, even if several entries for the same name can be found in the history (coming from multiple extrusions). Moreover, D is a largest such set. In consequence, several D sets can be extracted from the same transition, thus multiplying the number of causal transitions according to the different ways names could have been extruded. Note that the histories and processes are not changed by the above definition.

Proposition 2.19. *The causal semantics is more fine grained than the structural semantics of Fig. 1 in the sense that:*

1. *if* $(\overline{H}, \underline{H}) \vdash P \xrightarrow[u]{\alpha} (\overline{H}', \underline{H}') \vdash P'$ *then* $\exists D \subset \mathcal{H}$ *s.t.* $(\overline{H}, \underline{H}) \vdash P \xrightarrow[u,D]{\alpha} (\overline{H}', \underline{H}') \vdash P'$*;*
2. *if* $(\overline{H}, \underline{H}) \vdash P \xrightarrow[u,D]{\alpha} (\overline{H}', \underline{H}') \vdash P'$ *then* $(\overline{H}, \underline{H}) \vdash P \xrightarrow[u]{\alpha} (\overline{H}', \underline{H}') \vdash P'$ *was derived using the rules of Fig. 1.*

Proof. Trivial (note that D may be $\emptyset$). □

The link dependencies D allow us to define the final notion of events and independence relation for the causal semantics.

Definition 2.20 *(Causal independence).* Let the set of *events* **Ev** be defined by:

$$\mathbf{Ev} = \{\big((\alpha, u), D\big) \in Ev \times \mathcal{H} \mid \exists P, P', \overline{H}, \underline{H}, \overline{H}', \underline{H}' : (\overline{H}, \underline{H}) \vdash P \xrightarrow[u,D]{\alpha} (\overline{H}', \underline{H}') \vdash P'\}$$

Two events $e_i = (e_i', D_i) \in \mathbf{Ev}$ for $e_i' = (\alpha_i, u_i)$ and $i \in \{0, 1\}$ are independent, written $e_0 I e_1$, iff

$$e_0' I_s e_1' \;\wedge\; \nexists n : D_i(n) = u_{1-i} \quad \text{for } i \in \{0, 1\}.$$

Note that from the irreflexivity of the structural independence relation I_s, two events (e, D) and (e, D') that only differ on the splitting sets, i.e. $D \neq D'$, can never be independent.

We now briefly summarise the rules in Fig. 1.

The (IN) rule is the standard early input rule, substituting a received name n for the parameter x in P, yielding $P' = P\{x := n\}$, and enriched by recording the location $u = [\underline{a}(x).P][P']$ on the transition. Moreover, the rule takes care to add the name n to the input history $\underline{H}$.

The (OUT) rule is the standard output rule, enriched by bookkeeping the location $u = [\overline{a}\langle n\rangle.P][P']$ on the transition.

The (OPEN) rule is standard, except the location is recorded for the extruded name n in the output history and not in the action label, as is customary for the standard pi-semantics. Avoiding name extrusions in the labels ensures unique labels for events in Section 4, and only one (COM) rule. Otherwise, if we were to follow the pi-calculus conventions and have different action labels we would need to complicate the technicalities by adding a notion of equivalence of events (which would equate action labels with and without bound names), and proving results such as the independence relation has to respect this event equivalence (this more complicated approach can be seen in the technical report [35]).

The (SCOPE), (REP) and (SUM) rules are the standard rules, just extended to retain locations and histories.

If we do not consider the locations and histories, the (PAR$_i$) rules, for $i \in \{0, 1\}$, are the standard left and right parallel rules, except that we need to extract the possibly extruded name from the histories by the set $\tilde{b} = \mathrm{dom}(\overline{H}'_i) \backslash \mathrm{dom}([\check{i}]\overline{H})$, and not from the action label α. The location label is extended by prefixing with $i \in \{0, 1\}$, to record in which component of the parallel composition the action happened. The extruders recorded in the set $\overline{H}''$ capture exactly the parallel extrusion illustrated in Example 2.14. Specifically, an output location is added to the output extruder history, if the name has been extruded in the other parallel component and not previously extruded (recorded in the output history) nor received (recorded in the input history) by the current component.

Finally, if we again ignore histories and locations, the (COM$_i$) rules are the usual communication rules combined with the close rule, closing a scope previously opened by an (OPEN) rule. The combination is by abuse of notation, writing $(\nu\emptyset)P$ for P when there is the communication of a free name (in the same style to how the standard (CLOSE) rule for communication of a bound name). The location label is made into a pair, recording the two prefixes taking part in the communication. Looking at the histories, we discard any changes to histories formed in each component and only forward input histories from the sender to the receiver via the set $\underline{H}''$.

3. Correctness results and discussion of difference between late and early semantics

In this section we prove two correctness results for our semantics and in the end we remark on the difference between late and early semantics. The first correctness result shows that the standard interleaving, early operational semantics of pi-calculus recalled in Definition 2.2 can be obtained from the rules in Fig. 1 by ignoring the locations (Lemma 3.5) and histories (Lemma 3.2) and extracting only the scope extrusion from the histories. The result is stated as a bisimulation Corollary 3.8 from the two results of Proposition 3.6 which shows that every transition from pi-calculus is preserved in our semantics, and Proposition 3.7 which shows that our semantics does not introduce more transitions. The second correctness result from Theorem 3.9 shows that our semantics is a conservative extension of the one for CCS given in [33].

Definition 3.1 *(Generated transition system).* The operational rules for pi-calculus that we gave in Fig. 1 generate a transition system $TS = (S, E, \mathcal{T})$ in the following way.

- The set of *states* contains pairs of a history and a process term, denoted $(\overline{H}, \underline{H}) \vdash P$:

$$S \triangleq \{(\overline{H}, \underline{H}) \vdash P \mid P \in \mathbf{Proc}, (\overline{H}, \underline{H}) \in \mathcal{H} \times \mathcal{H}\},$$

  with **Proc** as in Definition 2.1 and histories from Definition 2.7;
- $E = \mathbf{Ev}$ are the events from Definition 2.20 which label the transitions;
- $\mathcal{T} \subseteq S \times E \times S$ are the transitions from Definition 2.18.

For some particular process P we add an initial state $(\emptyset, \emptyset) \vdash P$ and work only with the transition system *reachable* from this initial state, denoted $TS(P)$, and defined as the restriction $TS|_{S_P}$ of TS to only the states

$$S_P = \{(\overline{H}, \underline{H}) \vdash P' \mid (\emptyset, \emptyset) \vdash P \rightarrow^* (\overline{H}, \underline{H}) \vdash P'\},$$

with $\rightarrow^*$ being the reflexive and transitive closure of the transition relation $\mathcal{T}$.

We prove that in our semantics the histories do not affect the enabling of a standard π transition. The statement of Lemma 3.2 is on transitions directly derived from our semantic rules of Fig. 1 but it extends to causal semantics transitions (i.e., split transitions) as in Definition 2.18 due to Proposition 2.19.

Lemma 3.2 *(Histories in semantics). For a process P, if there exists a history $(\overline{H}, \underline{H})$ such that there exists a transition $(\overline{H}, \underline{H}) \vdash P \xrightarrow[u]{\alpha} (\overline{H}', \underline{H}') \vdash P'$ then for any history $(\overline{H}_v, \underline{H}_v)$ we can find the same transition $(\overline{H}_v, \underline{H}_v) \vdash P \xrightarrow[u]{\alpha} (\overline{H}_v', \underline{H}_v') \vdash P'$.*

Proof. We use induction on the length of the derivation tree for the given transition. We look at the last rule application (the root of the derivation tree), and show that we can apply it also to the different histories $(\overline{H}_v, \underline{H}_v)$.

The base cases for rules (IN) and (OUT) are easy since the histories are not influencing the decision of the transition (neither the action label nor the location label). The (IN) rule updates the resulting input history though, but this has no bearing on our lemma.

We take now cases.

1. For rules (REP), (SUM), and (SCOPE), we see that a general history $H = (\overline{H}, \underline{H})$ is propagated, unchanged, from the conclusion of the rule to the hypotheses. This allows to apply the induction hypothesis thus allowing to replace the history with any history $(\overline{H}_v, \underline{H}_v)$. Moreover, the adjacent requirements of these rules (when any) do not involve the history information.
2. For rule (OPEN) the left-side history is just propagated to the hypothesis, and the adjacent requirements do not involve the history. This allows to apply the induction hypothesis as before. The output history is changed though, but of no importance here.
3. For rules (PAR$_i$) the argument uses the induction hypothesis and also the fact that we can derive one transition with some history $(\overline{H}, \underline{H})$ from the process $P_0 \,||\, P_1$. We prove that we can derive the same transition from any other history and the same process $(\overline{H}_v, \underline{H}_v) \vdash P_0 \,||\, P_1$. From the fact that we can derive one transition it means we can derive from the component process $([\check{i}]\overline{H}, [\check{i}]\underline{H}) \vdash P_i \xrightarrow[u]{\alpha} (\overline{H}_i', \underline{H}_i') \vdash P_i'$. By the induction hypothesis this means we can derive the same transition from any output history, including the empty history as well as the history $[\check{i}]\overline{H}_v$. This means that for the outcome of the lemma we have the existence of the transition that is required by the (PAR$_i$). The other requirements of this rule do not influence its application, with one exception. In particular, the construction of $\overline{H}''$ is only used to update the output history on the right side of the derived transition.
   The exception is the fact that we need to check that no names from the other component process appear in the output history changes (i.e., in $\tilde{b} = \mathsf{dom}(\overline{H}_i''')\backslash\mathsf{dom}([\check{i}]\overline{H}_v)$). The intuition of $\tilde{b}$ is that it keeps the names that were opened, which in standard pi-calculus are called bound names in the action label. However, these bound action names are derived only from the process, and the history does not influence them. In other words, the changes in the domain of the output histories that $\tilde{b}$ records are the same for any history. Therefore, since we already know from the existing transition that $(\mathsf{dom}(\overline{H}_i')\backslash\mathsf{dom}([\check{i}]\overline{H})) \cap fn(P_j) = \emptyset$ this would hold for our arbitrary history as well $\mathsf{dom}([\check{i}]\overline{H}_v)$. Formally this is proven using induction on the derivation as the following Lemma 3.3.
4. The same argument as for (PAR) works for (COM) rules, only that we need to apply two times the induction hypothesis. □

Intuitively, the proof of Lemma 3.2 makes clear how only the process term is needed when deciding the existence of a transition, and the histories are only used to keep track of which names have been extruded, so that the (COM$_i$) rules can close the scope if needed. For the standard pi-calculus this is kept in the action label as the bound name of the action. Otherwise, histories are used in Definition 2.18 to split transitions.

The following two lemmas prove some simple observations which are used in several places throughout the paper, sometimes without explicit mention. Lemma 3.3 intuitively states that change in the domains of the output histories in a transition is the same irrespective of the left-side histories.

Lemma 3.3. *Consider a transition $(\overline{H}, \underline{H}) \vdash P \xrightarrow[u]{\alpha} (\overline{H}', \underline{H}') \vdash P'$ derived with the rules from Fig. 1. For any histories $\overline{H}_1, \overline{H}_1', \underline{H}_1, \underline{H}_1'$ s.t. the same transition is possible, i.e., $(\overline{H}_1, \underline{H}_1) \vdash P \xrightarrow[u]{\alpha} (\overline{H}_1', \underline{H}_1') \vdash P'$ we have* $\mathsf{dom}(\overline{H}) \setminus \mathsf{dom}(\overline{H}') = \mathsf{dom}(\overline{H}_1) \setminus \mathsf{dom}(\overline{H}_1')$.

Proof. We use an inductive argument based on the derivation tree for the transition, and look at the last rule application in cases.

The base cases are for (IN) and (OUT) for which the output histories are not changed by the transition, thus proving the claim trivially.

For the induction cases (REP), (SUM), and (SCOPE), just apply the induction hypothesis trivially, since the histories are copied from the hypothesis of the rule.

The case for (OPEN) changes the output history by adding the name n. However, since this name was coming from under a restriction operator (ν) it means the name cannot appear in any other component of the process. Therefore, the name n cannot appear in the left-hand history either; and for any other history we have to the left it would not contain n either, when this rule is applicable.

The case for $(\textsc{par}_i)$ is more complex since this transition changes the output histories in two ways. First, names at the current location are replaced (i.e., $[i]\overline{H}$ are removed) with the ones that the component transition outputs (i.e., $i\overline{H}_i'$ are added). This respects the claim by applying the induction hypothesis, which says that $\tilde{b} = \mathsf{dom}(\overline{H}_i') \setminus \mathsf{dom}([\check{i}]\overline{H})$ remains the same irrespective of $\overline{H}$; therefore, the same holds for $\mathsf{dom}(i\overline{H}_i') \setminus \mathsf{dom}([i]\overline{H})$, which is the same set of pairs as $\tilde{b}$ only with the locations prefixed by i, hence having the same domain. Second, it adds pairs in $i\overline{H}''$ which contains only those names that have previously been in the $\mathsf{dom}(\overline{H})$ because of the restriction $n \in \mathsf{dom}(i\overline{H}'')$ if $n \in \mathsf{dom}([j]\overline{H}) \subseteq \mathsf{dom}(\overline{H})$.

The case for $(\textsc{com}_i)$ is trivial, as the output histories are copied from the left to the right side of the conclusion ignoring what they might have been in the hypothesis. □

Lemma 3.4. *For any rule from Fig. 1 and the transition that it derives* $(\overline{H}, \underline{H}) \vdash P \xrightarrow[u]{\alpha} (\overline{H}', \underline{H}') \vdash P'$, *if for the left side of the transition we have* $\mathsf{dom}(\overline{H} \cup \underline{H}) \cap bn(P) = \emptyset$ *(ensured by the requirement in the caption of Fig. 1) then we have* $\mathsf{dom}(\overline{H}' \cup \underline{H}') \cap bn(P') = \emptyset$ *for the right side of the transition as well.*

Proof. The proof is done by induction on the derivation tree.

The base case for $(\textsc{out})$ rule is trivial since this does not change the histories. Whereas for the $(\textsc{in})$ rule the input history is changed by adding the name that is received. But this is fine since this name cannot appear bound in neither the process before nor the one after.

The induction case for the rules $(\textsc{scope})$, $(\textsc{rep})$, and $(\textsc{sum})$ is immediate by the induction hypothesis since the histories below the derivation line are the same as the ones above.

The $(\textsc{open})$ rule adds to the output histories the name n but removes the restriction operator on this name from the process before the transition. The induction hypothesis then finishes the proof.

For the $(\textsc{com})$ rules we need to prove that

$$\mathsf{dom}(\overline{H} \cup \underline{H} \cup j\underline{H}'') \cap (bn(P_0' \,||\, P_1') \cup \tilde{b}) = \emptyset,$$

knowing that $\mathsf{dom}(\overline{H} \cup \underline{H}) \cap bn(P_0 \,||\, P_1) = \emptyset$. Since $\underline{H}''$ contains only names that appear in $[i](\overline{H} \cup \underline{H})$ we thus have $\mathsf{dom}(\overline{H} \cup \underline{H} \cup j\underline{H}'') = \mathsf{dom}(\overline{H} \cup \underline{H})$. The requirement $\mathsf{dom}(\overline{H} \cup \underline{H}) \cap bn(P_0 \,||\, P_1) = \emptyset$ also implies that $\mathsf{dom}([\check{i}]\overline{H} \cup [\check{i}]\underline{H}) \cap bn(P_i) = \emptyset$, which allows to apply the induction hypothesis to derive also $\mathsf{dom}(\overline{H}_i' \cup \underline{H}_i') \cap bn(P_i') = \emptyset$. Since $\tilde{b} = \mathsf{dom}(\overline{H}_i') \setminus \mathsf{dom}([\check{i}]\overline{H})$, this also means that $\tilde{b} \cap bn(P_i') = \emptyset$ and that $\tilde{b}$ is fresh for the histories $\overline{H} \cup \underline{H}$. This means that (A) $\mathsf{dom}(\overline{H} \cup \underline{H} \cup j\underline{H}'') \cap \tilde{b} = \emptyset$. Since $bn(P_0') \subseteq bn(P_0)$ and $bn(P_1') \subseteq bn(P_1)$ we also have (B) $\mathsf{dom}(\overline{H} \cup \underline{H} \cup j\underline{H}'') \cap bn(P_0' \,||\, P_1') = \emptyset$. Thus, from (A) and (B) we have the proof.

Take one rule $(\textsc{par}_0)$, as the argument is analogous for the other, and we prove that $\mathsf{dom}\big(((\overline{H} \setminus [0]\overline{H}) \cup 0\overline{H}_0' \cup 0\overline{H}'' \cup (\underline{H} \setminus [0]\underline{H}) \cup 0\underline{H}_0')\big) \cap bn(P_0' \,||\, P_1) = \emptyset$. We take each set from the sequence of unions. For $\overline{H} \setminus [0]\overline{H}$ since it removes all names starting with 0 this will not contain bound names from P_0' either. From the assumption on the left of the transition, we know that the remaining names are not in P_1 either. The same argument goes for $\underline{H} \setminus [0]\underline{H}$. Now use the induction hypothesis to deduce that $\overline{H}_0' \cap bn(P_0') = \emptyset$ as well as $\underline{H}_0' \cap bn(P_0') = \emptyset$. Note that adding the digit 0 keeps the domains the same. We are left with $\overline{H}''$ which we know it is a subset of $[1]\underline{H}$ and does not already appear to have been part of the histories coming from the P_0 component. Therefore, by the assumption we also have that $\overline{H}''$ does not have names in the bound names of neither P_1 nor in P_0'. □

In order to compare our semantics with the standard one for pi-calculus we still need to show how the location labels can be ignored, since these can be determined solely from the two process terms involved in the transition.

Lemma 3.5 *(Location labels are determined). Whenever we have a transition* $(\overline{H}, \underline{H}) \vdash P \xrightarrow[u]{\alpha} (\overline{H}', \underline{H}') \vdash P'$ *then the location label u is unique, i.e., determined by P and P'.*

Proof. We generate the location u in the derivation tree of the transition. We assume inductively that the shorter tree generates a unique label. We then look at the structure of P and P', and consider cases on the last rule applied at the root of the derivation tree.

- If $P = \overline{a}\langle n\rangle.P_0'$ or $P = \underline{a}(n).P_0'$ then we know from $(\textsc{out})$ respectively $(\textsc{in})$ that $u = [\overline{a}\langle n\rangle.P_0'][P_0']$ respectively $u = [\underline{a}(n).P_0'][P_0']$ are unique. This forms the base case of the induction.
- If $P = P_0 \,||\, P_1$ and $P' = P_0' \,||\, P_1$ then we know that $(\textsc{par}_0)$ rule is applied at the root and it adds to the location label $u = 0v$ where v is the unique label obtained from the proof of the P_0 branch. Similarly we find $u = 1v$ if $P' = P_0 \,||\, P_1'$ using $(\textsc{par}_1)$.
- If $P = P_0 \,||\, P_1$ and $P' = P_0' \,||\, P_1'$ we know that $(\textsc{com})$ rule is applied and that $u = \langle 0u_0, 1u_1\rangle$, which is uniquely obtained by continuing up each branch finding the unique u_0 for the P_0 part and u_1 for the P_1 part.

- If P is of any other process type then the respective application of the remaining rules just copy the location labels. □

Proposition 3.6 *(Pi-calculus semantics is preserved). For a pi-process P whenever we have a transition $P \xrightarrow{\alpha}_{\pi} P'$ then we have in our semantics a transition $(\emptyset, \emptyset) \vdash P \xrightarrow[u]{\alpha'} (\overline{H}, \underline{H}) \vdash P'$ for some determined location label u and history $(\overline{H}, \underline{H})$, and when $\alpha = (\nu n)\overline{m}\langle n\rangle$ then $\alpha' = \overline{m}\langle n\rangle$ otherwise $\alpha' = \alpha$.*

Note that Proposition 3.6 claims the transition starting with the empty history, but then Lemma 3.2 allows any histories to be used. Lemma 3.5 tells how the location label u is determined.

Proof. We use induction on the derivation tree and show how for each rule application in the pi-calculus we find rules applied in our semantics which return the required process.

It is easy to see how when the pi-rules (INP) and (OUT) are applicable, then the respective rules in Fig. 1 are also applicable to the empty histories, and the output process as well as the action label are the same as in the pi-rule. In the rules from Fig. 1 we update the input history in the (IN) rule, and create the respective location labels.

The pi-rule (BANG) is the same as our rule (REP). The same for the pi-rule (RES) with its requirements which appear the same in our rule (SCOPE). The pi-rule (SUM) is as our rule. To all these the resulting processes are exactly the same. Moreover, the histories are not even changed, neither the location labels.

The pi-rules (PAR), (COM), and (OPEN) are manipulating the same processes as our respective rules from Fig. 1, and the respective requirements can be correlated. In our case we manipulate histories and construct location labels.

In particular, the (OPEN) rule from Fig. 1 has the same requirement on the name $n \neq a$ and it changes the output history to record the bound name that was output from under a scope, thus opening it. In the pi-rule this bound name is recorded in the action label, which in their case changes to $(\nu n)\overline{a}\langle n\rangle$. This is why the statement of this proposition captures this distinction between action labels.

For the pi-rules (PAR) the requirement $bn(\alpha) \cap fn(Q) = \emptyset$ is captured in our (PAR$_i$) rules by $\tilde{b} \cap fn(P_j) = \emptyset$ where $\tilde{b}$ is stored in the output history.

For the pi-rule (COM) note that the output action has no bound name, and thus no restriction operator is surrounding the parallel process. In our (COM) rules this implies that $\tilde{b} = \emptyset$, and we use the syntactic sugar $(\nu\emptyset)P_0 \,||\, P_1$ instead.

Otherwise, for the pi-rule (CLOSE) in our rules the $\tilde{b} \neq \emptyset$ and thus the restriction operator will be exactly as in the pi-rule.

One last aspect that we need to show is that the histories do not prohibit transitions which otherwise are allowed in the pi-calculus transition system. This was done in Lemma 3.2. □

Proposition 3.7 *(No extra transitions). For a process P whenever we have a transition in our semantics $(\overline{H}, \underline{H}) \vdash P \xrightarrow[u]{\alpha} (\overline{H}', \underline{H}') \vdash P'$ then we find the transition in the standard pi-calculus semantics $P \xrightarrow{\alpha'}_{\pi} P'$ with $\alpha' = (\nu n)\alpha$ if $\mathsf{dom}(\overline{H}')\backslash\mathsf{dom}(\overline{H}) = \{n\}$ and $\alpha' = \alpha$ otherwise.*

Proof. Consider our operational rules from Fig. 1 and use induction on the derivation tree to show that for each rule applied in our semantics when we remove the histories and location labels than either the same rule is applicable in the pi-calculus or we find other rules applied which return the required process. The reasoning is very similar to what we did in the proof of Proposition 3.6.

The only special aspect is the fact that we need to extract from the output history the action label when this needs to have a bound name, as coming from the (OPEN) rule. □

Combining Propositions 3.6 and 3.7 we now get the following corollary stating the conformance of our semantics with the standard early pi-semantics.

Corollary 3.8 *(Conformance w.r.t. pi-calculus). For any process P the generated transition system reachable from this process in the standard pi-semantics $TS_{\pi}(P)$ and our semantics $TS(P)$ are bisimilar in the following sense. There exists a relation R containing pairs of the form $(P', H \vdash P')$, where $H = (\overline{H}, \underline{H})$ is an extrusion history and such that*

- $(P, (\emptyset, \emptyset) \vdash P) \in R$
- *If $(P, H \vdash P) \in R$ and $\exists \alpha, P' : P \xrightarrow{\alpha}_{\pi} P'$ then $\exists H' : H \vdash P \xrightarrow[u]{\alpha'} H' \vdash P'$ where $\alpha' = \overline{m}\langle n\rangle$ if $\alpha = (\nu n)\overline{m}\langle n\rangle$ and otherwise $\alpha' = \alpha$.*
- *If $\exists H', P' : H \vdash P \xrightarrow[u]{\alpha} H' \vdash P'$ then there exists $P \xrightarrow{\alpha'}_{\pi} P'$ where $\alpha' = (\nu n)\alpha$ if $H = (\overline{H}, \underline{H})$, $H' = (\overline{H}', \underline{H}')$ and $\mathsf{dom}(\overline{H}')\backslash\mathsf{dom}(\overline{H}) = \{n\}$ and otherwise $\alpha' = \alpha$.*

We now show that the non-interleaving semantics is a conservative extension of the one for CCS given in [33]. To this end, consider as equivalent to CCS the sub-calculus of the pi-calculus obtained by allowing only input and output prefixes in which the subject and object are the same, i.e. of the form $\underline{n}(n)$ and $\overline{n}\langle n\rangle$, referred to as the CCS subset. In this case it is easy to see that the output histories and link dependencies D are always empty and thus the independence relation and events coincide with the structural independence and events.

Theorem 3.9 *(Conservative extension). For the CCS subset of the pi-calculus, the non-interleaving semantics of Fig. 1 is bisimilar to the non-interleaving semantics for CCS given in [33].*

Proof. To be precise, the CCS calculus from [33] builds processes using the grammar:

$$P^{CCS} ::= \Sigma_{i\in I}\varphi_i.P_i^{CCS} \mid (\nu n)P^{CCS} \mid P^{CCS} \,||\, Q^{CCS} \mid x \mid rec\,x.P^{CCS} \qquad \text{with } \varphi \in \{\overline{\alpha}, \underline{\alpha} \mid \alpha \in \mathcal{N}\}.$$

They use recursive definitions instead of our replication construct; but these are encodable, e.g., see [2, Def. 6], so we will not be concerned with this. Otherwise, the difference is in their action prefixes which are only names of the two kinds output/input. Therefore, this forms a subset of the pi-calculus that we considered in Definition 2.1 when we allow only output/input with the same subject and object, i.e., $\overline{n}\langle n\rangle \setminus \underline{n}(n)$. See a nice comparison of CCS and pi-calculus in this sense as psi-calculi instances in [4]. The proof is then formed of the following observations.

1. the output histories are always empty;
2. the link dependencies D are always empty, thus events are never split;
3. the independence relation and events coincide with the structural independence and events;
4. any transition in the CCS semantics is matched by the same transition in our semantics of Fig. 1, and the other way around.

To prove these we investigate our semantic rules, and discuss how these relate to the semantic rules from [33], which we do not reproduce here as they are revealed from our comparative arguments.

1. For the first statement we look at the transition rules (PAR_i) and (OPEN), as these are the only rules that add to the output histories. The rule (IN) adds to the input history, which is used in the (PAR_i) rules to determine what to add to the output histories. The (OPEN) rule is not applicable since it requires for an output prefix $\overline{a}\langle n\rangle$ that $n \neq a$, which does not hold for the prefixes that are allowed in the CCS instance. The extra names added in the (PAR_i) rules depend on previously added names by the (OPEN) rule, i.e., see the condition for forming the $\overline{H}''$ set in (PAR_i) rules. Since these are empty, then (PAR_i) rules do not add any new names either.
2. Since output histories are empty, and in Definition 2.18 the link dependencies set D is created from the histories as $D \subseteq \overline{H}$, this statement is trivially proven.
3. This is easy to see from the previous part 2 since Ds are always empty, then in Definition 2.20 an event $\big((\alpha, u), D\big) = (\alpha, u)$ as in Definition 2.10. Also, the split transition from which the causal events are extracted $(\overline{H}, \underline{H}) \vdash P \xrightarrow[u,\emptyset]{\alpha} (\overline{H}', \underline{H}') \vdash P'$ is the same as the transitions $(\overline{H}, \underline{H}) \vdash P \xrightarrow[u]{\alpha} (\overline{H}', \underline{H}') \vdash P'$ given by our semantic rules of Fig. 1. As the link dependencies are empty, then the second part of the definition of independence relation in Definition 2.20 is trivial, thus I becoming just the structural independence I_s as in Definition 2.12, which is that of CCS from [33].
4. We already observed that (OPEN) is never applicable, which implies that the "close" part of the (COM_i) rules is not applicable (i.e., when the $\tilde{b} \neq \emptyset$ and thus a restriction operation is moved from the parallel component with the output prefix to the whole parallel composition process). Otherwise, the remaining (COM_i) rules are the same as in [33], building the same location label, since our restriction about names is vacuously satisfied for CCS.
   Also note that the (IN) and (OUT) rules are always applicable as the immediate rule on top of the (SUM) rule. Therefore, we can view these three rules as the single SUM rule from [33] which is also performing the action, and exhibiting the respective output/input action name on the transition. However, the location label that we build in this case contains only the branch process, whereas in [33] it contains the whole summation process. This is fine for our proofs, and does not interfere with the deduced transitions.
   The (SCOPE) rule is the same as the RES rule from [33], also with the required restriction, only that in the location label we do not need to record the name that is bound by the restriction operator as [33] needs.
   For the (PAR_i) rules our restriction on names is vacuously satisfied, and the location label is updated exactly as in [33]. We do not discuss the (REP) rule.
   The above observations make it clear that whenever from a P^{CCS} process one can derive a transition in the semantics of [33], we can also derive the same transition, with the same action label, and analogous location label in our semantics. The location label in our case does not contain bound names, and contains in the last process part of the label only the summation branch. However, these location labels do not contribute in [33] to deciding which transitions can be derived. The opposite is also true, i.e., in our semantics we cannot derive a transition which the semantics of [33] cannot derive also. □

Remark 3.10 *(Late version of our semantics).* The transition for the input prefix generated by the (IN) rule in the standard early semantics (see Definition 2.2) contains a concrete name m, which is substituted for the parameter x. Assuming that all bound names are unique and different from all free names, one can define the transition for the input prefix generated by the (IN) rule in the late semantics simply by removing the substitution. The resulting process P will thereby contain the name x as a fresh free name. In the late semantics, the substitution is then performed in the (COM) rule. A late style version of the rules in Fig. 1 will be obtained by replacing the (IN) and (COM$_i$) by the following rules (for clarity we only give the (COM$_0$-LATE) rule replacing (COM$_0$)):

$$\frac{u = [\underline{a}(x).P][P]}{(\overline{H}, \underline{H}) \vdash \underline{a}(x).P \xrightarrow[u]{\underline{a}(x)} (\overline{H}, \underline{H}) \vdash P}\ (\text{IN-LATE})$$

$$\frac{\begin{array}{cc} \underline{H}'' = \{(n, v_1) \mid & \tilde{b} = \mathsf{dom}(\overline{H}'_0) \backslash \mathsf{dom}([\check{0}]\overline{H}) \\ \exists (n, l) \in [0](\overline{H} \cup \underline{H}) : l|_{\{0,1\}} \prec v_0\} & \tilde{b} \cap fn(P_1) = \emptyset \\ ([\check{0}]\overline{H}, [\check{0}]\underline{H}) \vdash P_0 \xrightarrow[v_0]{\overline{a}\langle n\rangle} (\overline{H}'_0, \underline{H}'_0) \vdash P'_0 & ([\check{1}]\overline{H}, [\check{1}]\underline{H}) \vdash P_1 \xrightarrow[v_1]{\underline{a}(x)} (\overline{H}'_1, \underline{H}'_1) \vdash P'_1 \end{array}}{(\overline{H}, \underline{H}) \vdash P_0 \,||\, P_1 \xrightarrow[\langle 0v_0, 1v_1\rangle]{\tau} (\overline{H}, \underline{H} \cup 1\underline{H}'') \vdash (\nu\tilde{b})(P'_0 \,||\, P'_1\{x := n\})}\ (\text{COM}_0\text{-LATE})$$

Because of the assumption that all bound names are unique and different from all free names, the received name x in the input transition with label $\underline{a}(x)$ will not be identical to any of the names that the extruders output. In consequence, we do not need to record the received name in the input history in the (IN-LATE) rule. Also, the case in Example 2.16 where the previously extruded name is received can not happen in the late semantics.

However, the input histories are still needed to keep track of names that are being received during communications. A new variant of Example 2.16 shows this and how (COM$_0$-LATE) still updates the input histories. Consider the process $P = (\nu n)(\overline{a}\langle n\rangle.\overline{b}\langle n\rangle \,||\, \underline{b}(x).\overline{c}\langle x\rangle)$ for which after an output on channel a, the name n is communicated on channel b between the location 0 and location 1. We still want that in the remaining process $\mathbf{0} \,||\, \overline{c}\langle n\rangle$ the output of n on channel c is not an extruder. This is done as in the early semantics, using the input histories in the (PAR) rules and updating them in the (COM$_i$-LATE) rules. In the late semantics, also Example 2.15 does no longer apply. This is captured by changing the definition of the splitting of events in Definition 2.18 by defining $no(\alpha)$ as $no(\overline{n}\langle m\rangle) = \{n\}\backslash\{m\}$, $no(\underline{n}(x)) = \{n\}$, i.e. only the subject (channel name) is considered for link dependencies. No other changes are needed.

The late semantics would thus have less events for two reasons: Firstly, the (IN-LATE) rule will only generate one transition, for the one (fresh) input $\underline{n}(x)$, instead of the infinitely many transitions for each possible name m in the action $\underline{n}(m)$ produced by the (IN) rule. Secondly, the splitting of these events will only depend on the subject and not the object of the communication. We leave for future work to prove that the above semantics provides an asynchronous transition system conservatively generalising the non-interleaving late semantics and to study in detail the correspondence between the late semantics given here and the one in [17]. The two semantics are not identical, since the semantics in [17] does not split events and thus results in a non-stable semantics.

4. Early labelled asynchronous transition systems semantics of pi-calculus

In this section we show that the operational semantics, events and independence relation given for the pi-calculus in the previous section yields a labelled asynchronous[6] transition system (LATS) as defined in [3,40,47,22] and recalled in Definition 4.1. LATS are known to satisfy the stability property, that is, every event depends on a unique set of events, and unfold to standard labelled prime event structures [47, Ch. 7].

Definition 4.1. A *labelled asynchronous transition system* is a tuple $LATS = (S, i, E, \mathcal{T}, I, lab, \mathcal{A})$ where

- $(S, i, E, \mathcal{T})$ is a transition system with S the set of *states* and i an *initial state*, E a set of *events*, and $\mathcal{T} \subseteq S \times E \times S$ the *transition relation*;
- $lab : E \to \mathcal{A}$ is a *labelling* map from the set of events to the *action set* $\mathcal{A}$;
- $I \subseteq E \times E$ is an irreflexive, symmetric independence relation, satisfying:
  1. $e \in E \;\Rightarrow\; \exists s, s' \in S : (s, e, s') \in \mathcal{T}$;
  2. $(s, e, s') \in \mathcal{T} \wedge (s, e, s'') \in \mathcal{T} \;\Rightarrow\; s' = s''$;
  3. $e_1 I e_2 \wedge \{(s, e_1, s_1), (s, e_2, s_2)\} \subseteq \mathcal{T} \;\Rightarrow\; \exists s_3 : \{(s_1, e_2, s_3), (s_2, e_1, s_3)\} \subseteq \mathcal{T}$;
  4. $e_1 I e_2 \wedge \{(s, e_1, s_1), (s_1, e_2, s_3)\} \subseteq \mathcal{T} \;\Rightarrow\; \exists s_2 : \{(s, e_2, s_2), (s_2, e_1, s_3)\} \subseteq \mathcal{T}$.

[6] Asynchronous here refers to non-interleaving, not the style of communication.

The last two conditions ensure that independent events always form interleaving diamonds and imply *stability*, i.e. unique cause of events.

Theorem 4.2 *(LATS for pi). The semantic rules for the pi-calculus that we gave in Fig. 1 generate a labelled asynchronous transition system* $LATS(P) = (\mathbf{Proc}, (\emptyset, \emptyset) \vdash P, \mathbf{Ev}, \mathcal{T}, I, lab, \mathcal{A})$ *for a pi-process* P *where*

- $(\mathbf{Proc}, (\emptyset, \emptyset) \vdash P, \mathbf{Ev}, \mathcal{T})$ *is the generated transition system* $TS(P)$ *reachable from* P *as in Definition 3.1 with* **Proc** *from Definition 2.1, histories from Definition 2.7, events from Definition 2.20, and transitions from Definition 2.18;*
- I *is the relation from Definition 2.20,*
- $lab\big((\alpha, u), D\big) = \alpha$,
- $\alpha \in \mathcal{A}$ *is the set of labels generated by the grammar* $\alpha ::= \overline{a}\langle n \rangle \mid \underline{a}(n) \mid \tau$.

To prove this we need several intermediate results and definitions. The following lemma states that the transition system is event deterministic, i.e. that it satisfies property 2. of Definition 4.1.

Lemma 4.3 *(Event determinism). For any two transitions* $(\overline{H}, \underline{H}) \vdash P \xrightarrow[u,D]{\alpha} (\overline{H}', \underline{H}') \vdash P'$ *and* $(\overline{H}, \underline{H}) \vdash P \xrightarrow[u,D]{\alpha} (\overline{H}'', \underline{H}'') \vdash P''$ *then* $(\overline{H}', \underline{H}') = (\overline{H}'', \underline{H}'')$ *and* $P' = P''$.

Proof. The events that we are interested in are the ones in **Ev** obtained through splitting in Definition 2.20. But each split event is obtained from a transition derived with the rules from Fig. 1 and all splits make transitions between the same pairs of processes and histories. Therefore we are not interested in the splits but in the kernel event, which is formed of the first two elements of the three-tuple event, i.e., the action name and the location where the event appears. In consequence, we show that for some arbitrary event and process (together with some history), there is a unique operational rule that is applicable. This results in a unique transition, as in the statement of the lemma.

The histories do not contribute to the decision whether an operational rule is applicable or not; and neither do the histories influence how the resulting process looks like. These facts have been established in Proposition 3.2.

For the rest we use induction on the structure of the location label and the process P, thus identifying which of the operational rules applies for the particular reduction. The induction is double, depending on how the location label looks like, cf. Proposition 2.6, i.e., either of type $(\alpha, s[P_1][P_1'])$ or $(\tau, s\langle s_0[P_0][P_0'], s_1[P_1][P_1']\rangle)$. We have two base cases.

1. When $e = (\alpha, [P_1][P_1'])$ only rules (IN) or (OUT) are applicable. Depending on the structure of the process P, or for the same purpose, depending on α, only one of the two rules applies. The outcome is then determined, thus having a unique resulting process. The histories are also determined, i.e., for the (OUT) rule they are the same as on the left of the transition, whereas for the (IN) rule the input history is changed with the name from the action α.
2. When $e = (\tau, \langle 0[P_0][P_0'], 1[P_1][P_1']\rangle)$ (i.e., when the location label is a minimal communication location) only the (COM) rule applies, and the outcome process is unique with respect to the original process P. If $P = (\nu b)Q$ then the rule (OPEN) also applies and changes the output action to one having a bounded name b, and the resulting process is also restricted as $(\nu b)Q'$ with Q' determined by the communicating processes. Otherwise, if P is only a parallel composition, then the resulting process is not under a restriction.

For the induction case we consider that the event has the location

$$ss'[P_0][P_0'] \quad \text{or} \quad ss'\langle 0s_0[P_0][P_0'], 1s_1[P_1][P_1']\rangle$$

with $s \in \{0, 1\}$, and any of the s', s_0, s_1 may be empty. We take cases after s and for each case we look at the structure of P and α to determine the rule used. It turns out that each time only one rule applies. Moreover, each rule changes the histories from the left of the transition in a determined way, ensuring the history equality from the statement of the lemma.

1. For the case when $s = 0$ we have that
   (a) if $P = P_0 \,||\, P_1$ then the only rule that could have been applied to generate this transition is (PAR$_0$). From the induction hypothesis we know that the required transition for this rule, which applies to a shorter location label, produces the unique process P_0', which when composed in parallel produces a unique outcome process $P_0' \,||\, P_1$.
   (b) if $P = !P_1$ then only rule (REP) applies to produce $Q \,||\, !Q$. Further up the derivation tree one of the (PAR) rules apply. When $s = 0$ the (PAR$_0$) rule produces the unique $P' \,||\, !Q$.
   (c) if $P = (\nu n)P_1$ then
      i. if $\alpha = \overline{a}\langle n \rangle$ then only rule (OPEN) applies,
      ii. otherwise only (SCOPE) is applicable.

   Further up in the derivation tree we can apply the induction hypothesis on the same location label, but a smaller process (without the restriction operator), to produce a unique process, thus the whole derivation is unique.

2. For the case when $s = 1$ we have that
   - if $P = P_0 \,||\, P_1$ then only rule (PAR$_1$) applies, and an argument analogous to 1a goes through.
   - if $P = !P_1$ then use analogous arguments as for 1b producing the unique $Q \,||\, P'$ by the induction hypothesis.
   - if $P = (\nu n)P_1$ then use analogous arguments as for 1c.
3. For all location labels and action labels if $P = \Sigma_{i \in I} : \varphi_i.P_i$ then the unique rule application is (SUM). Even if two branches of the sum induce a transition with the same action name but different output process, these will be considered as different events because of the location label containing different processes at the end. Therefore, the nondeterminism no longer exists here. The induction hypothesis is applicable further up on the derivation tree to the smaller label as well as smaller process, to produce a unique outcome.

For the location label $ss'\langle 0s_0[P_0][P_0'], 1s_1[P_1][P_1']\rangle$ an inductive argument as before reduces the s and s' to the case when they are empty. In this case the only rule that could have created this label is the (COM) rule. This implies that $P = Q_0 \,||\, Q_1$. Using the previous case on Q_0 and location $s_0[P_0][P_0']$ we know that Q_0 uniquely reduces to Q_0', and similarly for Q_1 and the location $s_1[P_1][P_1']$. Thus $Q_0 \,||\, Q_1$ reduces uniquely to $Q_0' \,||\, Q_1' = P'$. □

To prove that the transition system from Theorem 4.2 satisfies the last two (diamond) properties of a labelled asynchronous transition system we follow the approach from [33].

The following partial function makes precise how a sequence $s \in \{0,1\}^*$ identifies a subprocess, called the *component*, in a process.

Definition 4.4 *(Components).* Define inductively the partial function

$$Comp : \{0,1\}^* \times \mathbf{Proc} \rightharpoonup \mathbf{Proc}$$

1. $Comp(\epsilon, P) = P$, when $P \neq\, !P_1$ and $P \neq (\nu n)P_1$ (and ϵ is the empty string)
2. $Comp(0s, P_0 \,||\, P_1) = Comp(s, P_0)$
3. $Comp(1s, P_0 \,||\, P_1) = Comp(s, P_1)$
4. $Comp(s, (\nu n)P) = Comp(s, P)$
5. $Comp(s, !P) = Comp(s, P \,||\, !P)$

Corollary 4.5. *For any $s, s' \in \{0,1\}^*$ and any process P, whenever Comp is defined, we have*

$$Comp(s, Comp(s', P)) = Comp(s's, P).$$

Proof. The proof follows from Definition 4.4 by induction on the structure of s'. The base case for $s' = \varepsilon$ is easy by using Definition 4.4(1); and when $P = !P_1$ or $P = (\nu n)P_1$ then just apply the respective definitions.

Take $s' = 0s_0'$ for which *Comp* is defined when P is one of $P_0 \,||\, P_1$, $(\nu n)P_0$, or $!P_0$. For $P = P_0 \,||\, P_1$ from Definition 4.4(2) we have that $Comp(0s_0', P_0 \,||\, P_1) = Comp(s_0', P_0)$, which implies that the left part of the lemma equality becomes $Comp(s, Comp(s_0', P_0))$. The right part becomes $Comp(s_0's, P_0)$ by applying Definition 4.4(2) as $Comp(0s_0's, P_0 \,||\, P_1) = Comp(s_0's, P_0)$. The equality of these last two formulas is given by the induction hypothesis. In the case when $P = !P_0$ then we first apply Definition 4.4(5) and then we follow the above reasoning verbatim. For the case for $P = (\nu n)P_0$ we first use Definition 4.4(4) then follow as above.

The case for $s' = 1s_1'$ is analogous, using Definition 4.4(3) for $P = P_0 \,||\, P_1$ and Definition 4.4(4) for $P = (\nu n)P$.

Note that no matter what the structure of s is, the *Comp* is defined when $P = !P_0$. This then reduces to a process where only the cases Definition 4.4(2) and (3) could further be applicable. □

From any transition we can deduce the transition in the immediate component responsible for the derivation.

Lemma 4.6 *(Decomposing transitions). For $s \in \{\varepsilon, 0, 1\}$ and $s' \in \{\varepsilon, 0, 1\}^*$ we have*

$$(\overline{H}, \underline{H}) \vdash P \xrightarrow[ss'u_\varepsilon]{\alpha} (\overline{H}', \underline{H}') \vdash P' \quad \Rightarrow \quad ([\check{s}]\overline{H}, [\check{s}]\underline{H}) \vdash Comp(s, P) \xrightarrow[s'u_\varepsilon]{\alpha} (\overline{H}'', \underline{H}'') \vdash Comp(s, P'),$$

with u_ε either $[P_u][P_u']$ or $\langle 0s_0s_0'[P_0][P_0'], 1s_1s_1'[P_1][P_1']\rangle$, and depending on the case we have the following extra properties:

1. *when $s = 0$ we have $(\mathrm{dom}(\overline{H}'') \backslash \mathrm{dom}([\check{s}]\overline{H})) \cap fn(Comp(1, P)) = \emptyset$;*
2. *when $s = 1$ we have $(\mathrm{dom}(\overline{H}'') \backslash \mathrm{dom}([\check{s}]\overline{H})) \cap fn(Comp(0, P)) = \emptyset$;*
3. *when $s = \varepsilon$ and $P = (\nu\tilde{n})P_1$ and $P_1 \neq (\nu\tilde{m})P_2$, for $\tilde{n}$ and $\tilde{m}$ non-empty,*[7] *we either have $(\tilde{n} \cap n(\alpha) = \emptyset$ or $(\exists b \in \tilde{n} : \alpha = \overline{a}\langle b\rangle$ with $a \notin \tilde{n})$.*

[7] By the assumption that all bound names are unique and different from all free names we can prove that all restrictions can be moved to the front of the process term. Thus we denote by $(\nu\tilde{n})$ the list of all bound names of P.

Proof. We take cases after $s \in \{\epsilon, 0, 1\}$.

- When $s = 0$ the rule that adds 0 to the location label is (PAR$_0$), therefore we work with the transition $(\overline{H}, \underline{H}) \vdash P_0 \,||\, P_1 \xrightarrow[ss'u_\epsilon]{\alpha} (\overline{H}', \underline{H}') \vdash P_0' \,||\, P_1$. The rule requires that transition $([\check{0}]\overline{H}, [\check{0}]\underline{H}) \vdash P_0 \xrightarrow[s'u_\epsilon]{\alpha} (\overline{H}'', \underline{H}'') \vdash P_0'$ exists and that $\tilde{b} \cap fn(P_1) = \emptyset$, with $\tilde{b} = \mathrm{dom}(\overline{H}'') \backslash \mathrm{dom}([\check{0}]\overline{H})$. Applying Definition 4.4 for deriving components, the above transform into the expected result, i.e.: $([\check{0}]\overline{H}, [\check{0}]\underline{H}) \vdash Comp(0, P_0 \,||\, P_1) \xrightarrow[s'u_\epsilon]{\alpha} (\overline{H}'', \underline{H}'') \vdash Comp(0, P_0' \,||\, P_1)$ with $(\mathrm{dom}(\overline{H}'') \backslash \mathrm{dom}([\check{0}]\overline{H})) \cap fn(Comp(1, P)) = \emptyset$.
  If $s = 0$ and $P = !P_s$ the application of the (REP) rule implies that we have a transition from $(\overline{H}, \underline{H}) \vdash P_s \,||\, !P_s \xrightarrow[ss'u_\varepsilon]{\alpha} (\overline{H}', \underline{H}') \vdash P'$ with the same action, history, and label as the given transition of the statement of the lemma. On this we can now use the (PAR$_0$) rule and apply the same argument as above to deduce the transition $([\check{0}]\overline{H}, [\check{0}]\underline{H}) \vdash Comp(s, P_s \,||\, !P_s) \xrightarrow[s'u_\varepsilon]{\alpha} (\overline{H}'', \underline{H}'') \vdash Comp(s, P')$. By Definition 4.4(5) we know that $Comp(s, P) = Comp(s, P_s \,||\, !P_s)$, which ends this case.
- When $s = 1$ we follow the same argument as above using (PAR$_1$) instead of (PAR$_0$).
- When $s = \epsilon$ we consider the three rules (OPEN), (SCOPE), and (REP), where we do not add anything to the location label. The structure of P and of α determine which rule is applicable. We omit the histories when obvious from the context in order to not clutter the text.
  - Consider the application of (OPEN) to process $P = (\nu n)P_0$ and action $\alpha = \overline{a}\langle n\rangle$ where $a \neq n$. We thus work with a transition $(\nu n)P_0 \xrightarrow[\epsilon s'u_\epsilon]{\alpha} P_0'$ for which the rule ensures the existence of the transition $P_0 \xrightarrow[s'u_\epsilon]{\overline{a}\langle n\rangle} P_0'$. As $Comp(\epsilon, (\nu n)P_0) = P_0$ and $Comp(\epsilon, P_0') = P_0'$ we get the expected transition $Comp(\epsilon, (\nu n)P_0) \xrightarrow[s'u_\epsilon]{\overline{a}\langle n\rangle} Comp(\epsilon, P_0')$.
  - Consider the application of (SCOPE) to a process $P = (\nu n)P_0$, thus working with a transition $(\nu n)P_0 \xrightarrow[\epsilon s'u_\epsilon]{\alpha} (\nu n)P_0'$. The rule ensures that $n \notin n(\alpha)$ and that we have a transition $P_0 \xrightarrow[s'u_\epsilon]{\alpha} P_0'$. Applying Definition 4.4 we get the expected transition $Comp(\epsilon, (\nu n)P_0) \xrightarrow[s'u_\epsilon]{\alpha} Comp(\epsilon, (\nu n)P_0')$.
  - Consider the application of (REP) to a process $P = !P_0$, thus working with a transition $!P_0 \xrightarrow[\epsilon s'u_\epsilon]{\alpha} P'$. The rule ensures the existence of the transition $P_0 \,||\, !P_0 \xrightarrow[s'u_\epsilon]{\alpha} P'$. As $Comp(\epsilon, !P_0) = Comp(\epsilon, P_0 \,||\, !P_0) = P_0 \,||\, !P_0$ and $Comp(\epsilon, P') = P'$, we get the expected transition $Comp(\epsilon, !P_0) \xrightarrow[s'u_\epsilon]{\alpha} Comp(\epsilon, P')$. □

Corollary 4.7. *Applying several times Lemma 4.6, we can extend s to be a string of location components: $s \in \{0, 1\}^*$.*

From any communication transition we can then recover the transitions in the components identified by the location labels.

Lemma 4.8 *(Decomposing communications). For location strings $s, s_0, s_1, s_0', s_1' \in \{0, 1\}^*$ the following holds*

$$\textit{if}\quad (\overline{H}, \underline{H}) \vdash P \xrightarrow[s\langle 0s_0s_0'[P_0][P_0'], 1s_1s_1'[P_1][P_1']\rangle]{\tau} (\overline{H}', \underline{H}') \vdash P' \quad \textit{then}$$

$$([\check{s_l}]\overline{H}, [\check{s_l}]\underline{H}) \vdash Comp(s_l, P) \xrightarrow[s_0'[P_0][P_0']]{\alpha} (\overline{H}'', \underline{H}'') \vdash Comp(s_l, P') \quad \textit{and}$$

$$([\check{s_r}]\overline{H}, [\check{s_r}]\underline{H}) \vdash Comp(s_r, P) \xrightarrow[s_1'[P_1][P_1']]{\overline{\alpha}} (\overline{H}''', \underline{H}''') \vdash Comp(s_r, P')$$

where $s_l = s0s_0$, $s_r = s1s_1$, and $\overline{\alpha}$ notation is defined as $\overline{\overline{a}\langle n\rangle} = \underline{a}(n)$ and $\overline{\underline{a}(n)} = \overline{a}\langle n\rangle$.

Proof. We first make use of Lemma 4.6 to simplify to only considering $s = \varepsilon$. Then we work with the labels $0s_0$ and $1s_1$ and make two arguments in parallel.

Therefore, we consider P of the form $P_0 \,||\, P_1$ and apply the (COM) rule at the root of the transition derivation tree. This implies the existence of the two transitions $([\check{0}]\overline{H}, [\check{0}]\underline{H}) \vdash P_0 \xrightarrow[s_0s_0'[P_0][P_1']]{\alpha} (\overline{H}_0', \underline{H}_0') \vdash P_0'$ and $([\check{1}]\overline{H}, [\check{1}]\underline{H}) \vdash P_1 \xrightarrow[s_1s_1'[P_1][P_1']]{\overline{\alpha}} (\overline{H}_1', \underline{H}_1') \vdash P_1'$, with $Comp(0, P) = P_0$ and $Comp(1, P) = P_1$. Moreover, α and $\overline{\alpha}$ must be of the forms $\overline{a}\langle m\rangle$ and $\underline{a}(m)$, and $P' = (\nu\tilde{b})(P_0' \,||\, P_1')$, with $\tilde{b} = \mathrm{dom}(\overline{H}_0') \backslash \mathrm{dom}([\check{0}]\overline{H})$. We know that the channel name a must be the same in both α's and the output name m is the same as the received name on the input side. Moreover, if $\tilde{b} \neq \epsilon$, the only time a name can be added to an output action label is by the (OPEN) rule, which opens the scope of b and thus we must have $m = b$.

Apply now Lemma 4.6 (in fact Corollary 4.7 when the labels are strings) to the two transitions to obtain

$$([\check{s_0}][\check{0}]\overline{H}, [\check{s_0}][\check{0}]\underline{H}) \vdash Comp(s_0, P_0) \xrightarrow[s_0'[P_0][P_1']]{\alpha} (\overline{H}'', \underline{H}'') \vdash Comp(s_0, P_0')$$

and

$$([\check{s_1}][\check{1}]\overline{H}, [\check{s_1}][\check{1}]\underline{H}) \vdash Comp(s_1, P_1) \xrightarrow[s_1'[P_1][P_1']]{\overline{\alpha}} (\overline{H}''', \underline{H}''') \vdash Comp(s_1, P_1'),$$

which by Corollary 4.5 are the same as $([\check{0s_0}]\overline{H}, [\check{0s_0}]\underline{H}) \vdash Comp(0s_0, P) \xrightarrow[s_0'[P_0][P_1']]{\alpha} (\overline{H}'', \underline{H}'') \vdash Comp(s_0, P_0')$ respectively $([\check{1s_1}]\overline{H}, [\check{1s_1}]\underline{H}) \vdash Comp(1s_1, P) \xrightarrow[s_1'[P_1][P_1']]{\overline{\alpha}} (\overline{H}''', \underline{H}''') \vdash Comp(s_1, P_1')$.

To see how the right components become the ones from the statement of the lemma apply Definition 4.4(4) to P' with the respective location labels $0s_0$ and $1s_1$, having e.g.,

$$Comp(0s_0, P') = Comp(0s_0, (\nu\tilde{n})(P_0' \,||\, P_1')) = Comp(0s_0, P_0' \,||\, P_1') = Comp(s_0, P_0'). \quad \square$$

Conversely, we can lift a transition from a component to the whole process, when it does not enter a communication.

Lemma 4.9 *(Composing transitions). For $s \in \{\varepsilon, 0, 1\}$ we have that*

$$\text{if} \quad ([\check{s}]\overline{H}, [\check{s}]\underline{H}) \vdash Comp(s, P) \xrightarrow[s'u_\varepsilon]{\alpha} (\overline{H}', \underline{H}') \vdash P_1'$$

with u_ε either $[P_u][P_u']$ or $\langle 0s_0s_0'[P_0][P_0'], 1s_1s_1'[P_1][P_1']\rangle$, then

$$(\overline{H}, \underline{H}) \vdash P \xrightarrow[ss'u_\varepsilon]{\alpha} (\overline{H}'', \underline{H}'') \vdash P' \text{ with } Comp(s, P') = P_1'$$

under the following restrictions:

1. *for $s = 0$ if $(\mathrm{dom}(\overline{H}')\backslash\mathrm{dom}([\check{0}]\overline{H})) \cap fn(Comp(1, P)) = \emptyset$;*
2. *for $s = 1$ if $(\mathrm{dom}(\overline{H}')\backslash\mathrm{dom}([\check{1}]\overline{H})) \cap fn(Comp(0, P)) = \emptyset$;*
3. *for $s = \varepsilon$ and $P = (\nu n)P_1$ if $n \notin n(\alpha)$ or ($\alpha = \overline{a}\langle n\rangle$ and $n \neq a$).*

Proof. We take cases after $s \in \{\varepsilon, 0, 1\}$.

1. When $s = 0$ it implies that $P = P_0 \,||\, Q$ since $Comp(0, P) = P_0$, otherwise is undefined or it enters under the ϵ cases below. The requirements for the rule (PAR$_0$) are satisfied by the restrictions of the lemma. Therefore, we have the expected transition $(\overline{H}, \underline{H}) \vdash P_0 \,||\, Q \xrightarrow[0s'u_\varepsilon]{\alpha} (\overline{H}'', \underline{H}'') \vdash P_0' \,||\, Q$ and $Comp(0, P_0' \,||\, Q) = P_0'$, where $(\overline{H}'', \underline{H}'')$ are updated according to the (PAR$_0$) rule, but this is unimportant for this lemma.
2. When $s = 1$ use an argument as for $s = 0$ where we use rule (PAR$_1$) instead.
3. When $s = \varepsilon$ we consider the two non-trivial cases for which $Comp$ is applicable, i.e., when either $P = !P_1$ or $P = (\nu n)P_1$.
   - For $P = !P_1$ we have that $Comp(\epsilon, !P_1) = Comp(\epsilon, P_1 \,||\, !P_1) = P_1 \,||\, !P_1$. Therefore, the transition given by the lemma is in fact $P_1 \,||\, !P_1 \xrightarrow[s'[P_0][P_0']]{\alpha} P_1'$ and we are allowed to apply the rule (REP) to obtain $!P_1 \xrightarrow[s'[P_0][P_0']]{\alpha} P_1'$ which is the transition we are looking for, i.e., having $s = \varepsilon$, $P' = P_1'$, and $Comp(\epsilon, P_1') = P_1'$. The histories do not change, so we omitted them.
   - For $P = (\nu n)P_1$ we have $Comp(\epsilon, (\nu n)P_1) = Comp(\epsilon, P_1) = P_1$. We take two cases:
     - where $n \notin n(\alpha)$ for which the only applicable rule is (SCOPE) where we have the transition $([\check{\epsilon}]\overline{H}, [\check{\epsilon}]\underline{H}) \vdash (\nu n)P_1 \xrightarrow[s'[P_1][P_0']]{\alpha} (\overline{H}', \underline{H}') \vdash (\nu n)P_1'$. We also have that $Comp(\epsilon, (\nu n)P_1') = Comp(\epsilon, P_1') = P_1'$ and $([\check{\epsilon}]\overline{H}, [\check{\epsilon}]\underline{H}) = (\overline{H}, \underline{H})$ which end this case.
     - when $\alpha = \overline{a}\langle n\rangle$ and $n \neq a$, the only rule that can work with this action label is the (OPEN) rule, which will give us the transition $([\check{\epsilon}]\overline{H}, [\check{\epsilon}]\underline{H}) \vdash (\nu n)P_1 \xrightarrow[s'[P_0][P_0']]{\alpha} (\overline{H}'', \underline{H}'') \vdash P_1'$ and as $Comp(\epsilon, P_1') = P_1'$ and $([\check{\epsilon}]\overline{H}, [\check{\epsilon}]\underline{H}) = (\overline{H}, \underline{H})$ the statement of the lemma is true. $\square$

Corollary 4.10. *Applying several times Lemma 4.9, when the needed restrictions exist, we can extend s to be a string of location components: $s \in \{0, 1\}^*$.*

Lemma 4.11 *(Composing communications). Whenever we have*

$$([\check{0}]\overline{H}, [\check{0}]\underline{H}) \vdash Comp(0, P) \xrightarrow[s_0[P_0][P_0']]{\overline{a}\langle n\rangle} (\overline{H}_0', \underline{H}_0') \vdash P_0''$$

and

$$([\check{1}]\overline{H}, [\check{1}]\underline{H}) \vdash Comp(1, P) \xrightarrow[s_1[P_1][P_1']]{\underline{a}(n)} (\overline{H}_1', \underline{H}_1') \vdash P_1'',$$

for $a \notin \mathrm{dom}(\overline{H}_0' \setminus [\check{0}]\overline{H})$, then we have the communication

$$(\overline{H}, \underline{H}) \vdash P \xrightarrow[\langle 0s_0[P_0][P_0'], 1s_1[P_1][P_1']\rangle]{\tau} (\overline{H}', \underline{H}') \vdash P'$$

with $Comp(0, P') = P_0''$ and $Comp(1, P') = P_1''$. The symmetric case was elided.

Proof. The statement of the lemma implies that $Comp$ is applicable to P, both for a label 0 and 1. This means that P has the structure of a parallel composition. Because of the restriction $a \notin \tilde{b}$ we are allowed to use the (COM) rule, which gives us that $P' = (\nu\tilde{b})(P_0'' \,||\, P_1'')$ after the transition. From Definition 4.4(4) we have that $Comp(0, (\nu\tilde{b})(P_0'' \,||\, P_1'')) = Comp(s, (P_0'' \,||\, P_1''))$. which by Definition 4.4(2) becomes $Comp(0, P_0'' \,||\, P_1'') = P_0''$. Following the same reasoning using first Definition 4.4(4) and then Definition 4.4(3) we have that $Comp(1, P_0'' \,||\, P_1'') = P_1''$ proving the lemma. The histories are updated by the rule, but this is not relevant for the lemma, thus we denote them just $(\overline{H}', \underline{H}')$. The symmetric version follows the same lines of reasoning. □

Lemma 4.12 *(Localisation). For any process P and a location string s, whenever $Comp$ is defined, we have:*

1. *if $(\overline{H}, \underline{H}) \vdash P \xrightarrow[s[P_1][P_1']]{\alpha} (\overline{H}', \underline{H}') \vdash P'$ and $(s, s') \in I_l$ then $Comp(s', P) = Comp(s', P')$,*
2. *if $(\overline{H}, \underline{H}) \vdash P \xrightarrow[s\langle s_0[P_0][P_0'], s_1[P_1][P_1']\rangle]{\tau} (\overline{H}', \underline{H}') \vdash P'$ and $(ss_0, s') \in I_l$ and $(ss_1, s') \in I_l$ then $Comp(s', P) = Comp(s', P')$.*

Proof. We first prove the part 1.

Since $(s, s') \in I_l$ then we can split s and s' into $s = u0l_1$, $s' = u1l_1'$ respectively where $u = lcp(s, s')$ is the largest common prefix, or the symmetric $s = u1l_1, s' = u0l_1$, which can be treated analogous. From Corollary 4.5 we have that $Comp(s, P) = Comp(u0l_1, P) = Comp(0l_1, Comp(u, P))$, and denote $Comp(u, P) = P_u$. The decomposition Lemma 4.6 (i.e., Corollary 4.7 for u a string) allows us to derive the transition:

$$([\check{u}]\overline{H}, [\check{u}]\underline{H}) \vdash P_u = Comp(u, P) \xrightarrow[0l_1[P_\epsilon][P_\epsilon']]{\alpha'} (\overline{H}'', \underline{H}'') \vdash Comp(u, P') = P_u'$$

with $\alpha' = \alpha$ except for the case when (OPEN) rule is applied in the derivation tree, with $\alpha = \overline{a}\langle n\rangle$, $a \neq n$ and $\alpha' = \overline{a}\langle n\rangle$. Because of the location label $0l_1$ it means that $P_u = P_0 \,||\, P_1$, and applying Lemma 4.6 to the above transition we derive:

$$([\check{0}][\check{u}]\overline{H}, [\check{0}][\check{u}]\underline{H}) \vdash Comp(0, P_u) = P_0 \xrightarrow[l_1[P_\epsilon][P_\epsilon']]{\alpha'} (\overline{H}''', \underline{H}''') \vdash P_0' = Comp(0, P_u').$$

Lemma 4.6 also ensures that $(\mathrm{dom}(\overline{H}''')\setminus\mathrm{dom}([\check{u0}]\overline{H})) \cap fn(Comp(1, P_u)) = \emptyset$.

Moreover, since $(0, 1) \in I_l$ we can apply the current lemma inductively to the transition with location $0l_1$ to obtain that $Comp(1, P_u) = Comp(1, P_u')$. These (and the (PAR$_0$) rule that is applied to obtain the last transition) imply that $P_u' = P_0' \,||\, P_1$.

To finish the proof, use Corollary 4.5 to get $Comp(s', P) = Comp(1l_1', Comp(u, P))$, and we thus need to prove that

$$Comp(1l_1', Comp(u, P)) = Comp(l', Comp(u, P')).$$

We already know that $Comp(u, P) = P_u$ and $Comp(u, P') = P_u'$. Therefore, we need to show

$$Comp(1l_1', P_u) = Comp(1l_1', P_u').$$

Since we already have proven that $Comp(1, P_u) = Comp(1, P_u')$ this part is done.

For the part 2 we can consider ss_0 and s' to be written as $ss_0 = u_0l_0$ respectively $s' = u_0l_0'$, for which the independence says that u_0 is maximal and l_0 start with a 0 and l_0' with a 1 (the symmetric case is analogous). Similarly, for ss_1 and s' their independence implies that $ss_1 = u_1l_1$ and $s' = u_1l_1'$. Since $u_1l_1' = s' = u_0l_0'$ we can identify two cases:

i when $u_1 = u_0$ or

ii when $u_0 \prec u_1 \vee u_1 \prec u_0$.

Recall that since they are part of a communication location label the s_0 starts with a 0 and s_1 starts with a 1. The case (i) means that $u_0 \prec s \wedge u_1 \prec s$. Because of this, the component identified by s' is outside any of the two components that participate in the communication, and thus the rest of this case follows as for part 1.

For the case (ii) we know that ss_0 can be written as $s0s'_0$ and ss_1 can be written $s1s'_1$, which implies that $u_0 \prec u_1 \vee u_1 \prec u_0$. We work with the first substring inclusion, as the second one would be analogous. In this case we deduce that $u_0 = s$, for otherwise we would break the requirement that u_0 is maximal. This implies that s' can now be written as $s' = s1l''_0$ (when u_1 is the shortest then $s' = s0l''_1$), which means that the component we are working with is in the right part of the communication. Moreover, u_1 can be written as $s1v'$ and thus the whole $s' = s1v'l'_1$ and $ss_1 = s1v'l_1$. Since $s'I_lss_1$ and $s1v'$ is maximal then $l'_1 I_l l_1$ (and they start one with 0 and the other with 1).

Using the decomposition Lemma 4.8 we obtain the transition

$$([s\check{1}v']\overline{H}, [s\check{1}v']\underline{H}) \vdash Comp(s1v', P) \xrightarrow[l_1[R][R']]{\alpha} (\overline{H}''', \underline{H}''') \vdash Comp(s1v')P'.$$

We can apply the current lemma inductively to this transition and the independent labels $l'_1 I_l l_1$ to obtain

$$Comp(l'_1, Comp(s1v', P)) = Comp(l'_1, Comp(s1v')P')$$

which by Corollary 4.5 transforms in the expected result. □

Proof of Theorem 4.2. It is easy to see that the independence relation I of Definition 2.20 is irreflexive, because it inherits this from I_s, and symmetric, because I_s is and the second part is symmetric by definition.

For the first *ATS requirement 4.1(1)* each event in the generated transition system is a split event, which by Definition 2.18 is attached to a transition between two process.

The *ATS requirement 4.1(2)* is covered by Lemma 4.3.

To prove the *ATS requirement 4.1(3)* consider a history $(\overline{H}, \underline{H})$ and some process P, and two events $e = (\alpha, u, D), e' = (\alpha', u', D')$ enabled in this state $(\overline{H}, \underline{H}) \vdash P$, which are also independent eIe'. Take the two transitions corresponding to these events in the generated transition system, i.e.:

$$(\overline{H}, \underline{H}) \vdash P \xrightarrow[u,D]{\alpha} (\overline{H}', \underline{H}') \vdash P' \quad \text{and} \quad (\overline{H}, \underline{H}) \vdash P \xrightarrow[u',D']{\alpha'} (\overline{H}'', \underline{H}'') \vdash P''.$$

To satisfy 4.1(3) we need to prove the existence of the following transitions:

$$(\overline{H}', \underline{H}') \vdash P' \xrightarrow[u',D']{\alpha'} (\overline{H}''', \underline{H}''') \vdash P''' \quad \text{and} \quad (\overline{H}'', \underline{H}'') \vdash P'' \xrightarrow[u,D]{\alpha} (\overline{H}'''', \underline{H}'''') \vdash P''''$$

with $\overline{H}''' = \overline{H}'''', \underline{H}''' = \underline{H}'''', P''' = P''''$.

We can have three possible combinations of events, depending on the structure of their location labels:

1. $u = s[R_0][R'_0]$ and $u' = s'[R_1][R'_1]$;
2. $u = s\langle s_0[R_0][R'_0], s_1[R_1][R'_1]\rangle$ and $u' = s'[R_2][R'_2]$;
3. $u = s\langle s_0[R_0][R'_0], s_1[R_1][R'_1]\rangle$ and $u = s'\langle s_2[R_2][R'_2], s_3[R_3][R'_3]\rangle$.

We treat the first case.

Since eIe' then Definition 2.20 implies that eI_se' and that $\nexists n : D(n) = u' \wedge \nexists n : D'(n) = u$ (which we use towards the end of the proof, when showing the ATS Property 4.1(4)). Since $Loc(e) = \{s\}$ and $Loc(e') = \{s'\}$ we have by Definition 2.12 that sI_ls' which means that they look like $s = v0s_3$ and $s' = v1s'_3$ with $v = lcp(s, s')$ being the largest common prefix. In consequence, $Comp(v, P) = P_0 \,||\, P_1$ is a parallel composition, and the decomposition Lemma 4.6 applied using v allows to derive the transitions

$$([\check{v}]\overline{H}, [\check{v}]\underline{H}) \vdash Comp(v, P) \xrightarrow[0s_3[R_0][R'_0]]{\alpha} (\overline{H}'_v, \underline{H}'_v) \vdash Comp(v, P') \quad \text{and}$$

$$([\check{v}]\overline{H}, [\check{v}]\underline{H}) \vdash Comp(v, P) \xrightarrow[1s'_3[R_1][R'_1]]{\alpha'} (\overline{H}''_v, \underline{H}''_v) \vdash Comp(v, P'').$$

For this part of the proof we can ignore the D, D' sets because these are derived from the histories of transitions like the above. Therefore, we reason over simpler transitions as above, and in the end we argue that the same split transitions on the respective D and D' can be obtained from the resulting histories.

At this point the two derivation trees are different in the sense that in one the (PAR$_0$) rule is applicable whereas in the other the (PAR$_1$) rule. From the two (PAR) rules we have the following respective transitions

$$([\check{v0}]\overline{H},[\check{v0}]\underline{H}) \vdash P_0 \xrightarrow[s_3[R_0][R_0']]{\alpha} (\overline{H}_0',\underline{H}_0') \vdash P_0' \quad \text{and} \quad ([\check{v1}]\overline{H},[\check{v1}]\underline{H}) \vdash P_1 \xrightarrow[s_3'[R_1][R_1']]{\alpha'} (\overline{H}_1',\underline{H}_1') \vdash P_1'.$$

According to the decomposition Lemma 4.6 we also have $Comp(v0, P) = P_0$ and $Comp(v1, P) = P_1$, as well as $Comp(v0, P') = P_0'$ and $Comp(v1, P'') = P_1'$. Moreover, Lemma 4.6 also provides the following $\mathsf{dom}(\overline{H}_0') \setminus \mathsf{dom}([\check{v0}]\overline{H}) \cap fn(P_1) = \emptyset$ respectively $\mathsf{dom}(\overline{H}_1') \setminus \mathsf{dom}([\check{v1}]\overline{H}) \cap fn(P_0) = \emptyset$.

Now we show how to derive the first expected transition $(\overline{H}',\underline{H}') \vdash P' \xrightarrow[u',D']{\alpha'} (\overline{H}''',\underline{H}''') \vdash P'''$. Because of the localisation Lemma 4.12 applied to labels $0s_3I_l1$ we know that $P_1 = Comp(1, Comp(v, P)) = Comp(1, Comp(v, P'))$ and thus we can deduce that $Comp(v, P') = P_0' \,||\, P_1$. To this we can now apply the rule (PAR$_1$) with the right transition from above, and deduce the transition

$$(\overline{H}_v',\underline{H}_v') \vdash P_0' \,||\, P_1 = Comp(v, P') \xrightarrow[1s_3'[R_1][R_1']]{\alpha'} (\overline{H}_v''',\underline{H}_v''') \vdash P_0' \,||\, P_1'.$$

We were able to apply the right transition from above with a different (correct) history because of Lemma 3.2. Moreover, the requirement of the (PAR$_1$) rule (i.e., $\tilde{b} \cap fn(P_0') = \emptyset$, with $\tilde{b}$ the change in histories from the respective transition above) was provided above by Lemma 4.6 but through the result in Lemma 3.3, which says that the change in the domain of the output histories $\tilde{b} = \mathsf{dom}(\overline{H}_v''') \setminus \mathsf{dom}(\overline{H}_v')$ is the same for any starting histories (using moreover the fact that $fn(P_0) = fn(P_0')$).

In order to obtain the full required transition we need to show that now we can apply Lemma 4.9 for composing transitions on the above last transition (i.e., the respective restrictions of Lemma 4.9 need to be satisfied). For this we consider a minimal $v \in \{0, 1\}$ and take cases depending on it.

Consider $v = 0$ (or for $v = 1$ an analogous argument will go through), i.e., we use $s = 00s_3$ and $s' = 01s_3'$. Apply the localisation Lemma 4.12 with the independent locations $s = 00s_3I_l1$ to deduce that $Comp(1, P) = Comp(1, P')$, and therefore, $P' = P_0' \,||\, P_1 \,||\, Q_1$. The restriction for applying Lemma 4.9, i.e.: $(\mathsf{dom}(\overline{H}_0') \setminus \mathsf{dom}([\check{0}]\overline{H})) \cap fn(Comp(1, P') = Q_1) = \emptyset$, is given by the decomposition Lemma 4.6 when it was applied to the second transition with the above particular $v = 0$. This is because the lemma had to apply the (PAR$_0$) rule which keeps the restriction on names $(\mathsf{dom}(\overline{H}_0') \setminus \mathsf{dom}([\check{0}]\overline{H})) \cap fn(Comp(1, P))$ which is equal, as mentioned above, to Q_1.

To any of the above we can apply rules that do not change the location label, i.e. the (OPEN), (SCOPE), or (REP). The first two need the form of P to be $P = (\nu n)Q$, for which $Comp(v, P) = Q$ for any location label v. The change of histories will tell which of (OPEN) or (SCOPE) have been applied. For both these rules the decomposition Lemma 4.6(3) provides the restrictions needed by the composition Lemma 4.9(3).

In consequence, in all cases we can apply the composition Lemma 4.9 to $Comp(v, P') \xrightarrow[1s_3'[R_1][R_1']]{\alpha'} P_0' \,||\, P_1'$ to obtain $P' \xrightarrow[v1s_3'[R_1][R_1']]{\alpha'} P'''$ with $Comp(v, P''') = P_0' \,||\, P_1'$.

We can use analogous reasoning to obtain a transition $Comp(v, P'') \xrightarrow[0s_3[R_0][R_0']]{\alpha} P_0' \,||\, P_1'$. We can apply the composition Lemma 4.9 to obtain $P'' \xrightarrow[v0s_3[R_0][R_0']]{\alpha} P''''$ with $Comp(v, P'''') = P_0' \,||\, P_1'$.

It remains to argue that $P''' = P''''$. This is the case because of the localisation Lemma 4.12 which says that for any independent location label $v'I_lv$ we have:

- $Comp(v', P) = Comp(v', P')$ as well as $Comp(v', P) = Comp(v', P'')$ from the given transitions;
- $Comp(v', P') = Comp(v', P''')$ as well as $Comp(v', P'') = Comp(v', P'''')$ for the above deduced transitions.

Therefore, we get $Comp(v', P''') = Comp(v', P'''')$, which together with the fact that $Comp(v, P''') = P_0' \,||\, P_1' = Comp(v, P'''')$ we have our expected result $P''' = P''''$.

Claim: The histories H' and H'' are changed by the two derived transitions into the same history $H''' = H''''$.

We look in the derivation tree of the transition to identify how histories are being changed. Observe that rules (SUM), (REP), (SCOPE), copy the histories from the hypothesis to the conclusion, i.e., they only propagate down the tree the changes done further up by the other rules. Rule (OUT) makes no change to the histories.

We discuss how the rest of the rules (IN), (PAR$_i$), (OPEN), (COM$_i$), change the histories.

The only rule that adds new names to the input history $\underline{H}$ is (IN) and this can only be applied once in a derivation tree. Therefore, the action label α of the transition will tell whether the (IN) rule has been applied, and thus we know exactly what has been added to the input history, i.e., the pair (n, ε) with n the name from the action $\alpha = \underline{a}(n)$. Moreover, when (IN) is applied then in the same derivation tree the rule (OUT) cannot appear, and thus also not the rule (OPEN) as it needs further up in the tree an application of (OUT) (as seen from the action label in the hypothesis of (OPEN)). In consequence, in a derivation tree, together with rule (IN) only (PAR) or (COM) rules may still be applicable. The (PAR) rules update any history information coming from further up in the derivation tree by adding the respective component label, i.e., replacing

$[i]\overline{H}$ by $i\overline{H}'_i$ for extruder histories and $[i]\underline{H}$ by $i\underline{H}'_i$ for input histories. The (COM) rules update the input histories with the label of the component that does the input in the communication.

The (COM) rules, on the other hand, do not change the extruder histories (the same as e.g., (OUT) does not), since in the conclusion we have the same output histories both on the left and on the right of the transition. But further up in the derivation tree histories may change, in response to (OPEN) or (PAR_i).

The (PAR) rules add new information to the extruder history as $\overline{H}''$. The pair that is added in the $\overline{H}''$ depends on the action label $\alpha = \overline{a}\langle n\rangle$ and on the history that we work with (both input and extruder parts of the history). In any case, at most one name is added to the history. Moreover, when (IN) is applicable, then $\overline{H}''$ is empty because it depends on α being an output action.

All these tell that when the actions on the transitions that we work with, i.e., either α or α' are input actions, then the history is changed by adding one name pair to the input history $\underline{H}$ and leaving the output history unchanged.

The (OPEN) rule requires a previous application of (OUT), and therefore, no application of (IN) is possible, thus the input history $\underline{H}$ remains unchanged throughout the derivation tree. The (OPEN) rule adds one single pair (n, u) which we extract from the transition labels, i.e., having $\alpha = \overline{a}\langle n\rangle$ and the location label u. Note that at this point in the derivation tree the name n is added as extruder with the current location u, but this location is updated by the (PAR) rules through prefixing with location labels depending on the respective component. Thus, at the root of the tree this extruded name appears as new in the history but with the full label u which we see on the transition that we work with.

To finish the proof of this claim we take cases after α and α'.

1. When both $\alpha = \underline{a}(n)$ and $\alpha' = \underline{b}(m)$ are input actions.
   The given histories are $H' = H \cup H_1$ where $H_1 = (\emptyset, \{(n, u)\})$, and $H'' = H \cup H_2$ where $H_2 = (\emptyset, \{(m, u')\})$. Important is that adding new entries to the input histories does not depend on the previous history H. Therefore, when we apply the input actions in the derived transitions, i.e., α' to H' and α to H'', we add the respective input names. This means that $H''' = H' \cup H_2 = H \cup H_1 \cup H_2$ and $H'''' = H'' \cup H_1 = H \cup H_2 \cup H_1$, which are the same. This reasoning works even when some of the names a, x, n, b, y, m are equal.
2. When both $\alpha = \overline{a}\langle n\rangle$ and $\alpha' = \overline{b}\langle m\rangle$ are output actions, where both n, m can be extruded names or not, and not necessarily different.
   (a) Consider both actions extrude their name and $n \neq m$ (i.e., the initial process P contains both $(\nu n)(\nu m)$); then the given histories are $H' = H \cup H_1$ where $H_1 = (\{(n, u)\}, \emptyset)$, and $H'' = H \cup H_2$ where $H_2 = (\{(m, u')\}, \emptyset)$. Since both actions are initial extruders for their names, it means that the (OPEN) rule has been applied in both derivation trees. Whenever (OPEN) is applied then we are guaranteed that the name added to the extruder history does not already exist in the extruder history part, i.e., (OPEN) cannot be applied twice with the same name during the execution of a process.
   Therefore, if H_1 adds name n to H and H_2 adds name m, and neither of the names existed in H, then it is obvious that when in the deduced transitions the α' is applied to H' it will add H_2. The same for the other transition to obtain the expected result as in the previous case.
   (b) When both actions extrude the same name $n = m$ (and thus P contains only (νn)), we are in the situation of parallel extruders. Here is the case when in the initial transitions we apply the rule (OPEN) whereas in the derived transitions the rule (PAR) (for possible extruder). This is fine since the name n will be initially added to the histories H' and H'' by the (OPEN) rule, and then in the deduced transitions it will satisfy the requirements to be added to the histories again, but with an independent location.
   (c) The case when P does not have a (ν) but the history contains a name $n \in \overline{H}$ is handled by the (PAR) rules since this means that the respective output is a possible extruder (not an initial extruder). For the (PAR) rule to add to the extruder history, i.e., so $\overline{H}'' \neq \emptyset$, the name that needs to be added must already exist in the history but with an independent location label. Assume this for α, thus $(n, u'') \in \overline{H}$ with $u'' I_l u$. No matter what the α' does it only adds to the histories and thus we will still have $(n, u'') \in \overline{H}''$. Therefore α will still add the (n, u), and thus we get the same histories in both deduced transitions.
   (d) Any other possibilities are as simple as the last case above, or are trivial since they do not change any history (like when the outputs do not extrude).
3. When α is an input action and α' is an output action the reasoning is simple similar to the one done in the last cases.

Claim: The deduced transitions can be split with the respective D, D' based on the respective histories.

From the two given transitions we know that both $D \subseteq \overline{H}$ and $D' \subseteq \overline{H}$. Since the histories only (at most) increase through transitions and we have seen that $H \subseteq H''$ then we also have that $D \subseteq H''$, meaning that the transition can be split with D (the same for $D' \subseteq H'$).

For the second case, when locations $u = s\langle s_0[R_0][R'_0], s_1[R_1][R'_1]\rangle$ and $u' = s'[R_2][R'_2]$, the proof follows similar arguments as for the first case, with few differences which we comment about in the following. Note that the independence of the events now offers two pairs of independent locations: $ss_0 I_l s'$ and $ss_1 I_l s'$, which give rise to several cases. Recall that s_0 and s_1 start with a 0 respectively 1.

Assume that in both independences we have $v = lcp(ss_0, s') \prec s$ and $v' = lcp(ss_1, s') \prec s$, which means that $v = v'$. It is easy to see that now we can immediately apply the same reasoning as we did before, for the first case. Moreover, recall that the (com) rule which is applied for the transition with u does not change histories, and thus the transition at u does not change histories. This makes the above reasoning about histories even simpler and we can easily deduce the existence of the splitting sets D, D'.

Assume that $v = s$ which means that $ss_0 = v0s'_0$ and $s' = v1s'_3$. This in turn implies that $s'_3 \neq \varepsilon$ because otherwise we would get $s' \prec ss_1 = v1s'_1$ which breaks the independence of these two. Moreover, since $v1s'_1 = ss_1 I_l s' = v1s'_3$ we get that $s'_1 I_l s'_3$ and denote $lcp(s'_1, s'_3) = v''$, thus having $ss_1 = v1v''0s''_1$ and $s' = v1v''1s''_3$ (or a symmetric variant).

Since u is a communication location label the (com) rule must have been applied in the derivation tree, which requires $Comp(v, P) = P_0 \,||\, P_1$, thus, having the following transitions

$$(\overline{H}, \underline{H}) \vdash P \xrightarrow[v\langle 0s'_0[R_0][R'_0], 1v''0s''_1[R_1][R'_1]\rangle, D]{\tau} (\overline{H}', \underline{H}') \vdash P' \quad \text{and} \quad (\overline{H}, \underline{H}) \vdash P \xrightarrow[v1v''1s''_3[R_2][R'_2], D']{\alpha'} (\overline{H}'', \underline{H}'') \vdash P''.$$

We must deduce the existence of the following two transitions

$$(\overline{H}', \underline{H}') \vdash P' \xrightarrow[u', D']{\alpha'} (\overline{H}''', \underline{H}''') \vdash P''' \quad \text{and} \quad (\overline{H}'', \underline{H}'') \vdash P'' \xrightarrow[u, D]{\tau} (\overline{H}'''', \underline{H}'''') \vdash P'''',$$

with $\overline{H}''' = \overline{H}''''$, $\underline{H}''' = \underline{H}''''$ and $P''' = P''''$. We concentrate on deducing the left transition.

Using Lemma 4.6 on the first given transition we can find the following transition (ignoring the splitting sets D, D' for now, as we did before)

$$([\check{v}]\overline{H}, [\check{v}]\underline{H}) \vdash Comp(v, P) \xrightarrow[\langle 0s'_0[R_0][R'_0], 1v''0s''_1[R_1][R'_1]\rangle]{\tau} (\overline{H}'_v, \underline{H}'_v) \vdash Comp(v, P').$$

At this point the (com) rule is applicable and we know that $Comp(v, P) = P_0 \,||\, P_1$ to some $Comp(v, P') = P'_0 \,||\, P'_1$. Using the decomposition of communications Lemma 4.8 we get two transitions

$$([v\check{}0]\overline{H}, [v\check{}0]\underline{H}) \vdash Comp(0, Comp(v, P)) \xrightarrow[s'_0[R_0][R'_0]]{\alpha} (\overline{H}'_{v0}, \underline{H}'_{v0}) \vdash Comp(0, Comp(v, P')) \text{ and}$$

$$([v\check{}1]\overline{H}, [v\check{}1]\underline{H}) \vdash Comp(1, Comp(v, P)) \xrightarrow[v''0s''_1[R_1][R'_1]]{\overline{\alpha}} (\overline{H}'_{v1}, \underline{H}'_{v1}) \vdash Comp(1, Comp(v, P')),$$

with $(\overline{H}'_{v0}, \underline{H}'_{v0})$ and $(\overline{H}'_{v1}, \underline{H}'_{v1})$ some arbitrary histories (the indexes are only meant to help keep track where they come from). Applying more the decomposition Lemma 4.6 and Corollary 4.5 to the last transition we obtain

$$([v1\check{}v'']\overline{H}, [v1\check{}v'']\underline{H}) \vdash Comp(v1v'', P) \xrightarrow[0s''_1[R_1][R'_1]]{\overline{\alpha}} (\overline{H}'_{v1v''}, \underline{H}'_{v1v''}) \vdash Comp(v1v'', P').$$

Whereas applying the same decomposition Lemma 4.6 to the second given transition from the beginning we obtain

$$([v1\check{}v'']\overline{H}, [v1\check{}v'']\underline{H}) \vdash Comp(v1v'', P) \xrightarrow[1s''_3[R_2][R'_2]]{\alpha'} (\overline{H}''_{v1v''}, \underline{H}''_{v1v''}) \vdash Comp(v1v'', P'').$$

Because of the localisation Lemma 4.12 and the independence of these last two location labels we can apply the same transition with α' but to the component of P', i.e., deduce

$$Comp(v1v'', P') \xrightarrow[1s''_3[R_2][R'_2]]{\alpha'} Comp(v1v'', P'').$$

Note that the histories do not matter at this point, because of the same arguments that we gave before; therefore we do not mention histories in these last transitions. We can lift this transition using localisation Lemma 4.12 and the composition of transitions Lemma 4.9 because the restrictions are satisfied by the previous decomposition lemma applications (similarly as we argued in the first case above) and obtain the transition

$$Comp(v, P') \xrightarrow[1v''1s''_3[R_2][R'_2]]{\alpha'} Comp(v, P'').$$

Because of the independence with the label $0s'_0$ we can apply the localisation lemma to deduce that this part of the process also remains unchanged. Therefore, when applying again the composition Lemma 4.9 we lift the transition to the top-most process, as expected

$$P' \xrightarrow[v1v''1s''_3[R_2][R'_2]]{\alpha'} P'''$$

with P''' the same as P' only with the component $Comp(v1v'', P')$ changed accordingly. This component is different than the part that was changed through the communication.

To deduce the second transition apply a similar argument, but when lifting up apply the composition of communications Lemma 4.11 to deduce the correct τ-transition. The resulting process will be the same as P''' because when first we applied the transition with α' we change a part which was then not touched by the communication transition.

It is easy to see that these deduced transitions are allowed, and that the histories are the same, i.e., $H''' = H'''' = H''$.

For the fourth ATS requirement of Definition 4.1(4) consider a history H, some process P, an event $e = (\alpha, u, L)$ with a transition $H, P \xrightarrow[u]{[\alpha],L} H', P'$, and an event $e' = (\alpha', u', L')$ with a transition $H', P' \xrightarrow[u']{[\alpha'],L} H'', P''$, where we also have that eIe'. To satisfy 4.1(4) we need to prove the existence of the following transitions

$$H, P \xrightarrow[u']{[\alpha'],L'} H''', P''' \quad \text{and} \qquad H''', P''' \xrightarrow[u]{[\alpha],L} H'', P''.$$

We will only prove the first of the transitions and rely on the proof for the requirement Definition 4.1(3) to show the existence of the second transition. This is the case because once we have deduced the transition from P, deducing the fourth transition will fall in under Definition 4.1(3).

We have four different combinations of events depending on the structure of their location labels:

1. $u = s[R_0][R'_0]$ and $u' = s'[R_1][R'_1]$;
2. $u = s\langle s_0[R_0][R'_0], s_1[R_1][R'_1]\rangle$ and $u' = s'[R_2][R'_2]$;
3. $u = s[R_0][R'_0]$ and $u' = s'\langle s_1[R_1][R'_1], s_2[R_2][R'_2]\rangle$;
4. $u = s\langle s_0[R_0][R'_0], s_1[R_1][R'_1]\rangle$ and $u = s'\langle s_2[R_2][R'_2], s_3[R_3][R'_3]\rangle$.

For case 1 we first write s and s' as we did in the proof of case 1 for 4.1(3) as $s = lcp(s, s')0v$ and $s' = lcp(s, s')1v'$ (the symmetric versions follow just the same). Using the decomposition Lemma 4.6 on the two given transitions we have the two transitions

$$Comp(lcp(s, s'), P) \xrightarrow[0v]{\alpha} Comp(lcp(s, s'), P')$$

and, coming after it,

$$Comp(lcp(s, s'), P') \xrightarrow[1v']{\alpha'} Comp(lcp(s, s'), P'').$$

We want to prove the existence of the transition $Comp(lcp(s, s'), P) \xrightarrow[1v']{\alpha'} P'''_0$. Having this we can employ the composition Lemma 4.9, because its restrictions are offered by the previous decomposition lemma, similar to what we argued before, and we deduce the required transition $P \xrightarrow[lcp(s,s')1v']{\alpha'} P'''$ with $Comp(lcp(s, s'), P''') = P'''_0$.

Since $0vI_l1v'$ we can apply the localisation Lemma 4.12 to the first transition to deduce that

$$Comp(1v', Comp(lcp(s, s'), P)) = Comp(1v', Comp(lcp(s, s'), P')).$$

In consequence we can apply the second transition to this component to obtain the transition that we are looking for, with $P'''_0 = Comp(lcp(s, s'), P'')$. Here we have to apply the decomposition and composition lemmas to go down respectively up the location string, similar to what we argued in the previous relevant case for ATS restriction 4.1(3).

It remains to make sure that the above derived transition is allowed, which by Definition 3.1 means to show that $L' \subseteq \overline{H}$. We know that $L' \subseteq \overline{H}'$ and we know that $H' = H \cup H_0$ where H_0 may be of the form $(\{n, u|_{\{0,1,[\mathbf{P}]\}}\}, \emptyset)$. In fact, any name that is added to $\overline{H}_0$ by α will have the location $u|_{\{0,1,[\mathbf{P}]\}}$. To show our inclusion we can show that $L' \cap \overline{H}_0 = \emptyset$. This is given by the independence relation $eI_\pi e'$ which implies that $\nexists n : L'(n) = u|_{\{0,1,[\mathbf{P}]\}}$.

The remaining three cases follow similar arguments as the above case 1 and the respective cases 2 and 3 from the proof for the ATS requirement 4.1(3). □

5. Conclusion and related work

We provided the first stable, non-interleaving, early operational semantics for the pi-calculus conservatively generalising the interleaving early operational semantics. The semantics is given as labelled asynchronous transition systems (LATS). We followed and conservatively generalised the approach for CCS in [33] by capturing the link causalities introduced by the pi-calculus processes through a notion of extrusion histories inspired from the recent work of [17]. In this respect, our semantics treats all the examples that are discussed in [17, Sec. 6].

Part of our motivation was to give a non-interleaving semantics that is close to the standard early semantics of pi-calculus; hence our use of operational rules and a transition-like model. Moreover, we aimed to stay close to standard non-interleaving models; hence the stable model of asynchronous transition systems, which required us to split events in order to handle disjunctive causalities.

There have been several works on causal models for pi-calculi using various techniques. The work of [12,13] uses indexed labelled asynchronous transition systems (N-LATS), which extend the general works based on category theory for CCS [3,47], looking at early bisimulation semantics [5,6]. This work takes what they refer to as a model-theoretic approach, identifying N-LATS as a transition system model suitable for representing causality and linking of pi-calculi independently of the syntax. This work uses similar concepts as the work of [17] that we were inspired by; i.e., the N-LATS keep a set of names, and use special labels and dependencies between these labels in order to keep track of the special link dependencies that pi-calculus introduces. However, [13] views the model categorically, whereas [17] not; and [13] works with an extension of LATS whereas [17] with a similar extension of event structures. A deeper investigation of the relationship between our semantics and the one in [12,13] could perhaps lead to also a deeper understanding between our semantics and the one of [17].

The work of [24] takes a domain theoretic semantic of the pi-calculus given in terms of pomsets and appropriate operations on these. The papers [9,21] present Petri nets semantics for the pi-calculus, and thus, the same as the previous work, are not structural operational in nature.

The work of [6] makes the distinction between “subject and object dependencies”, which we also adopted, and they define the notion of causal bisimulation to distinguish processes wrt. their subject or object dependencies. However, this causal equivalence is shown to be encoded into the ordinary interleaving observational equivalence, thus not giving the true concurrency semantics that we are looking for in this paper. A similar study is carried by [38] in the setting of location bisimulation [8] with a similar (fully abstract) encoding into observational bisimulation as above. Good comparative overviews of location equivalence versus other causality equivalences are provided by [26,15]. This last line of work is what we adopt in our paper as well, i.e., using locations (static locations, as opposed to dynamic locations as above) to capture where in a process the concurrent actions are performed, i.e., by which parallel component. We started from the work of [33] for CCS; however there exist several related works including [1,10] for static locations and [8] for dynamically assigned locations. The handbook chapter [11] gives a good introduction and more recent overview of this subject. A thorough investigation of non-interleaving semantics for CCS is carried in [7], which compared to [33] and our work, uses the general choice operator instead of guarded summation. In our case we would need to enrich the labels to keep track of the branches being taken, similar to what [7] are doing. However, [7] records the whole computation along with the structure of the choices (which should not be confused with what we record in our extruder histories); which might be too much for our case, and an approach as in [11] with sequences of L/R branching markers could be enough. The models used in [7] (i.e., using equivalences of sequences of transitions) capture what LATS capture using the diamond properties. The paper [20] also investigates the distinction between the CCS causality and the specific pi-calculus causalities as [6] did, but which they call “structural and link dependencies”. The paper studies to some extent the notion of extruders generating different kinds of causalities in the late style of pi-calculus. They also make use of the same static location labels as in the case of CCS above, which they call “proof terms”.

None of these approaches deal with the notion of parallel extruders, which implies disjunctive causalities and thus requires event splitting in order to regain stability of the concurrency models. The work of [32] introduces a graph rewriting semantics of the early style of pi-calculus (without summation) that deals with parallel extruders. More thoroughly, the works of [16,45,17] treat parallel extruders and full pi-calculus in a denotational semantics, using an extended form of event structure with a set holding the bound names of the process.

The work of [19] on reversible pi-calculus (Rpi) also provides a stable causal semantics as a by-product, given in [19, Chap. V] when one views Rpi as an annotated version of pi-calculus. Concretely, the memories in Rpi carry more information (needed for the purpose of reversibility) than needed for capturing causalities (i.e., information of extrusion that we record in histories). Still, it is not trivial to provide a formal relationship between their memories and our histories, mainly because we use explicit location labels, whereas they use a split symbol on the memory stacks. It could also be worth noting that (to simplify presentation) the syntax used in [19] does not consider choices nor replication (i.e., only covers finite, deterministic processes). However, [19] does not explicitly identify the events and thus neither a non-interleaving (event-based) model underlying their semantics (as we do here using asynchronous transition systems). The reversible semantics is given in terms of reduction contexts, with no given formal relations with other models. Note also that a main goal of [19] is to have a compositional semantics, whereas we paid attention to obtaining a stable operational model.

As future work we aim to prove that the late semantics given in Remark 3.10 yields a labelled asynchronous transition system, and then explore the differences between early and late style non-interleaving semantics in the setting of this paper. We conjecture that it is possible to define a splitting for the extended event structures of [17] to obtain the prime event structures given by unfolding the asynchronous transition systems provided by our late semantics. This we believe would also be relevant for comparing the semantics of Rpi [19] with our semantics and the compositional denotational semantics of [17]. An alternative could be to work with more expressive models of concurrency than prime event structures and ATS, which can also capture disjunctive causalities, like higher dimensional automata [36,44] or configuration structures [43] and ST-structures [25] if aiming for an event-based model.

The free choice (compared to our use of guarded choice in this paper) introduces more details in a non-interleaving semantics; see [11] or our report [35] where we have worked out the generalisation of the present semantics to unguarded choice. Finally, we are working on lifting the present results to *psi-calculi* [4], laying the foundations for generalising the meta-theory and operational semantics of psi-calculi to non-interleaving semantics.

Acknowledgements

We are grateful for suggestions and comments from anonymous reviewers of earlier versions of this work.

References

[1] L. Aceto, A static view of localities, Form. Asp. Comput. 6 (2) (1994) 201–222.

[2] J. Aranda, C.D. Giusto, C. Palamidessi, F.D. Valencia, On recursion, replication and scope mechanisms in process calculi, in: F.S. de Boer, M.M. Bonsangue, S. Graf, W.P. de Roever (Eds.), Formal Methods for Components and Objects, FMCO'06, in: LNCS, vol. 4709, Springer, 2007, pp. 185–206.

[3] M.A. Bednarczyk, Categories of Asynchronous Systems, PhD thesis, Univ. Sussex, 1988.

[4] J. Bengtson, M. Johansson, J. Parrow, B. Victor, Psi-calculi: a framework for mobile processes with nominal data and logic, Log. Methods Comput. Sci. 7 (1) (2011).

[5] M. Boreale, D. Sangiorgi, A fully abstract semantics for causality in the pi-calculus, in: STACS, 1995, pp. 243–254.

[6] M. Boreale, D. Sangiorgi, A fully abstract semantics for causality in the pi-calculus, Acta Inform. 35 (5) (1998) 353–400.

[7] G. Boudol, I. Castellani, Flow models of distributed computations: three equivalent semantics for CCS, Inf. Comput. 114 (2) (1994) 247–314.

[8] G. Boudol, I. Castellani, M. Hennessy, A. Kiehn, A theory of processes with localities, Form. Asp. Comput. 6 (2) (1994) 165–200.

[9] N. Busi, R. Gorrieri, A Petri net semantics for pi-calculus, in: CONCUR, in: LNCS, vol. 962, Springer, 1995, pp. 145–159.

[10] I. Castellani, Observing distribution in processes: static and dynamic localities, Int. J. Found. Comput. Sci. 6 (4) (1995) 353–393.

[11] I. Castellani, Process algebras with localities, in: Handbook of Process Algebra, Elsevier, 2001, pp. 945–1045 (chapter 15).

[12] G.L. Cattani, P. Sewell, Models for name-passing processes: interleaving and causal, in: LICS, IEEE Computer Society, 2000, pp. 322–333.

[13] G.L. Cattani, P. Sewell, Models for name-passing processes: interleaving and causal, Inf. Comput. 190 (2) (2004) 136–178.

[14] E. Clarke, O. Grumberg, M. Minea, D. Peled, State space reduction using partial order techniques, Int. J. Softw. Tools Technol. Transf. 2 (3) (1999) 279–287.

[15] F. Corradini, R. De Nicola, Locality based semantics for process algebras, Acta Inform. 34 (4) (1997) 291–324.

[16] S. Crafa, D. Varacca, N. Yoshida, Compositional event structure semantics for the internal *pi*-calculus, in: CONCUR, in: LNCS, vol. 4703, Springer, 2007, pp. 317–332.

[17] S. Crafa, D. Varacca, N. Yoshida, Event structure semantics of parallel extrusion in the pi-calculus, in: FOSSACS, in: LNCS, vol. 7213, Springer, 2012, pp. 225–239.

[18] I. Cristescu, Operational and Denotational Semantics for the Reversible π-Calculus, PhD thesis, Université Paris Diderot – Paris 7 – Sorbonne Paris Cité, 2015.

[19] I. Cristescu, J. Krivine, D. Varacca, A compositional semantics for the reversible pi-calculus, in: ACM/IEEE Symposium on Logic in Computer Science, LICS, IEEE Computer Society, 2013, pp. 388–397.

[20] P. Degano, C. Priami, Non-interleaving semantics for mobile processes, Theor. Comput. Sci. 216 (1–2) (1999) 237–270.

[21] J. Engelfriet, A multiset semantics for the pi-calculus with replication, Theor. Comput. Sci. 153 (1&2) (1996) 65–94.

[22] T.T. Hildebrandt, Categorical Models for Concurrency: Independence, Fairness and Dataflow, PhD thesis, University of Aarhus, Denmark, 1999.

[23] T.T. Hildebrandt, C. Johansen, H. Normann, A stable non-interleaving early operational semantics for the pi-calculus, in: 11th International Conference on Language and Automata Theory and Applications, in: LNCS, vol. 10168, 2017, pp. 51–63.

[24] L.J. Jagadeesan, R. Jagadeesan, Causality and true concurrency: a data-flow analysis of the pi-calculus, in: AMAST, in: LNCS, vol. 936, Springer, 1995, pp. 277–291.

[25] C. Johansen ST-structures, J. Log. Algebraic Methods Program. 85 (6) (2016) 1201–1233.

[26] A. Kiehn, Comparing locality and causality based equivalences, Acta Inform. 31 (8) (1994) 697–718.

[27] I. Lanese, C.A. Mezzina, J.-B. Stefani, Reversibility in the higher-order pi-calculus, Theor. Comput. Sci. 625 (2016) 25–84.

[28] R. Milner, A Calculus of Communicating Systems, vol. 92, Springer-Verlag, 1980.

[29] R. Milner, Communicating and Mobile Systems: The π-Calculus, Cambridge Univ. Press, 1999.

[30] R. Milner, J. Parrow, D. Walker, A calculus of mobile processes, I–II, Inf. Comput. 100 (1) (1992) 1–77.

[31] R. Milner, J. Parrow, D. Walker, Modal logics for mobile processes, Theor. Comput. Sci. 114 (1) (1993) 149–171.

[32] U. Montanari, M. Pistore, Concurrent semantics for the pi-calculus, Electron. Notes Theor. Comput. Sci. 1 (1995) 411–429.

[33] M. Mukund, M. Nielsen, CCS location and asynchronous transition systems, in: FSTTCS, in: LNCS, vol. 652, Springer, 1992, pp. 328–341.

[34] M. Nielsen, G. Plotkin, G. Winskel, Petri nets, event structures and domains, in: Semantics of Concurrent Computation, in: LNCS, vol. 70, Springer, 1979, pp. 266–284.

[35] H. Normann, C. Johansen, T. Hildebrandt, Non-interleaving Operational Semantics for the Pi-Calculus (Long Version), Technical Report 453, Dept. Info., University of Oslo, 2016, http://heim.ifi.uio.no/~cristi/papers/TR453.pdf.

[36] V.R. Pratt, Modeling concurrency with geometry, in: POPL'91, 1991, pp. 311–322.

[37] P. Quaglia, The pi-calculus: notes on labelled semantics, Bull. Eur. Assoc. Theor. Comput. Sci. 68 (1999) 104–114.

[38] D. Sangiorgi, Locality and interleaving semantics in calculi for mobile processes, Theor. Comput. Sci. 155 (1) (1996) 39–83.

[39] D. Sangiorgi, D. Walker, The π-Calculus: A Theory of Mobile Processes, Cambridge Univ. Press, 2001.

[40] M.W. Shields, Concurrent machines, Comput. J. 28 (5) (1985) 449–465.

[41] I. Ulidowski, I. Phillips, S. Yuen, Concurrency and reversibility, in: Proceedings Reversible Computation: 6th International Conference, RC 2014, Kyoto, Japan, July 10–11, 2014, in: LNCS, vol. 8507, Springer, 2014, pp. 1–14.

[42] R. van Glabbeek, U. Goltz, Refinement of actions and equivalence notions for concurrent systems, Acta Inform. 37 (4/5) (2001) 229–327.

[43] R. van Glabbeek, G. Plotkin, Configuration structures, event structures and Petri nets, Theor. Comput. Sci. 410 (41) (2009) 4111–4159.

[44] R.J. van Glabbeek, On the expressiveness of higher dimensional automata, Theor. Comput. Sci. 356 (3) (2006) 265–290.

[45] D. Varacca, N. Yoshida, Typed event structures and the linear π-calculus, Theor. Comput. Sci. 411 (19) (2010) 1949–1973.

[46] G. Winskel, Events, causality and symmetry, Comput. J. 54 (1) (2011) 42–57.

[47] G. Winskel, M. Nielsen, Models for concurrency, in: S. Abramski, D. Gabbay, T. Maibaum (Eds.), Handbook of Logic in Computer Science, Oxford, 1995, pp. 1–148.