A Stable Non-interleaving Early Operational Semantics for the Pi-calculus (author version)*

Thomas Hildebrandt[1], Christian Johansen[2]**, and Håkon Normann[1]

[1] IT University of Copenhagen, Rued Langgaardsvej 7, 2300 Copenhagen, Denmark.
[2] Dept. Informatics, University of Oslo, P.O.Box 1080 Blindern, 0316 Oslo, Norway.

Abstract. We give the first non-interleaving early operational semantics for the pi-calculus which generalizes 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. The semantics is conservatively extended 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 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 [16] is the seminal model for concurrent mobile processes, representing mobility by the fresh creation and communication of channel names. The standard operational semantics adopt an *interleaving* approach to concurrency, that represent concurrent execution of actions as their arbitrary sequential interleaving and employ basic transition systems or automata as semantic models. However, the ability to distinguish concurrency from interleaving has several practical applications, including dealing with state-space explosion in model-checking [8], supporting action refinement [25] and reversibility (e.g. [24,14]).

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 [21,25] or employ a non-interleaving model (e.g. [19,26,1,23,5]), 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 [3], the source of the complexity is that the causal dependencies fall in two categories: 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 [17,13,5,3,21,7,9,11,10] and process algebras in general (e.g. [4,6]).

Among the most recent work, a stable operational semantics for reversible, deterministic and finite pi-calculus processes is provided in [11]. Stability means that every event depends on a unique history of past events, which supports reversibility of computations. A denotational semantics for the pi-calculus is provided in [9] 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. 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. 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.

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 [22]. Our starting point is the work of Mukund and Nielsen [18], which defines a structural operational non-interleaving semantics for

* !This is an author's produced version, and not the final one published by Springer, which can be found at http://dx.doi.org/10.1007/978-3-319-53733-7_3

** The second author (with previous name 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.

Milner's CCS [15] as (labelled) asynchronous transition systems using locations to identify the structural causality, which is the only type of causality in CCS. We generalize the approach of [18] to the pi-calculus by, in additon to the structural locations, employing a notion of *extruder histories*, recording the location of both name extrusions and name inputs. Together, the locations and extruder histories allow to identify the underlying events of transitions and both their structural and link causal dependencies.

Overview of paper: In Sec. 2 we generalise the structural operational early semantics for the pi-calculus with locations for transitions, extrusion histories and link dependencies. In Sec. 3 we show that the semantics yields a standard labelled asynchronous transitions system, which is known to unfold to labelled prime event structures [26]. We conclude and comment on future work in Sec. 4. Proof details can be found in the companion technical report [20].[3]

2 Causal Early Operational Semantics

In this section we give an early operational semantics of the pi-calculus recording both the structural and link causal dependencies between events.

We first recall the syntax for the pi-calculus with guarded choice.

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* n, m, *by the grammar:*

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

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. We assume all bound names are unique in a process and identify processes up to* α*-conversion.*

In Fig. 1 we give the causal early semantics for pi-processes with transitions of the form $(\overline{H}, \underline{H}) \vdash P \xrightarrow[u]{\alpha} (\overline{H}', \underline{H}') \vdash P'$. Following the approach in [18] we have added *location labels* u under the transitions, identifying the location of the prefixes in the term contributing to a transition and allowing to infer structural (CCS-like) events and causalities. To capture the finer notion of events and link causalities of the pi-calculus, we enrich the semantics with *extrusion histories* $(\overline{H}, \underline{H})$ to the left of the turnstile, which record the location in the parallel process of the prefixes extruding respectively receiving some name.

Formally, we define the set of prefix locations as follows.

Definition 2.2. *Let* $\mathcal{L} = \{0,1\}^* \times \mathbf{Proc} \times \mathbf{Proc}$ *be the set of* prefix locations *and write* $s[P][P']$ *for elements in* $\mathcal{L}$.

The location labels u of transitions in Fig. 1 are then of the following forms:

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

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 branch of a parallel composition, and P' being the residual sub-term after the transition. The location labels of the second form provide two prefix locations, $sis_i[P_i][P_i'] \in \mathcal{L}$ for $i \in \{0,1\}$, identifying the output and input prefix in a communication.

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.

From the locations we define our first notion of structural events and independence, which correspond to the events and independence defined for CCS in [18]. We will later show how to take into account the additional link causal relationships of the pi-calculus.

Definition 2.3. *Let* $Ev = \{(\alpha, u) \mid (\overline{H}, \underline{H}) \vdash P \xrightarrow[u]{\alpha} (\overline{H}', \underline{H}') \vdash P'\}$ *be the* structural events. *For* $e = (\alpha, u) \in Ev$ *define* $Loc(e) \subseteq \{0,1\}^*$, *the* locations *where* e *occurs, by*

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

[3] The report employs a slightly different and more complex, but equivalent exposition of the semantic rules.

$$\frac{u = [\underline{a}(m).P][P'] \quad P' = P[m := n]}{(\overline{H}, \underline{H}) \vdash \underline{a}(m).P \xrightarrow[u]{\underline{a}(n)} (\overline{H}, \underline{H} \cup \{(n,u)\}) \vdash P'} \text{(IN)}$$

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

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

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

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

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

$$\frac{\begin{array}{ll}\overline{H}'' = \{(n,u) \mid \alpha = \overline{a}\langle n\rangle, n \in \mathsf{dom}([j]\overline{H}), & P_j = P'_j \\ \quad \forall l.l|_{\{0,1\}} \prec iu : (n,l) \notin \overline{H} \cup \underline{H}\} & j = 1 - i \\ \quad ([\check{i}]\overline{H}, [\check{i}]\underline{H}) \vdash P_i \xrightarrow[u]{\alpha} (\overline{H}'_i, \underline{H}'_i) \vdash P'_i & \tilde{b} \cap n(P_j) = \emptyset\end{array}}{(\overline{H}, \underline{H}) \vdash P_0 \,||\, P_1 \xrightarrow[iu]{\alpha} \big((\overline{H} \backslash [i]\overline{H}) \cup i(\overline{H}'_i \cup \overline{H}''), (\underline{H} \backslash [i]\underline{H}) \cup i\underline{H}'_i\big) \vdash P'_0 \,||\, P'_1} \text{(PAR}_i\text{)}$$

$$\frac{\begin{array}{ll}\underline{H}'' = \{(n,v) \mid & \\ \exists (n,l) \in [i](\overline{H} \cup \underline{H}) : l|_{\{0,1\}} \prec u\} & \tilde{b} \cap n(P_j) = \emptyset, \quad j = 1 - i \\ ([\check{i}]\overline{H}, [\check{i}]\underline{H}) \vdash P_i \xrightarrow[u]{\overline{a}\langle n\rangle} (\overline{H}'_i, \underline{H}'_i) \vdash P'_i & ([\check{j}]\overline{H}, [\check{j}]\underline{H}) \vdash P_j \xrightarrow[v]{\underline{a}(n)} (\overline{H}'_j, \underline{H}'_j) \vdash P'_j\end{array}}{(\overline{H}, \underline{H}) \vdash P_0 \,||\, P_1 \xrightarrow[\langle 0u, 1v\rangle]{\tau} (\overline{H}, \underline{H} \cup j\underline{H}'') \vdash (\nu\tilde{b})(P'_0 \,||\, P'_1)} \text{(COM}_i\text{)}$$

Fig. 1. Early operational semantics enriched with *action labels* $\alpha ::= \tau \mid \overline{a}\langle n\rangle \mid \underline{a}(x)$, *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 that for all rules, if $(\overline{H}, \underline{H}) \vdash P \xrightarrow[u]{\alpha} (\overline{H}', \underline{H}') \vdash P'$ is the conclusion, we require $\mathsf{dom}(\overline{H} \cup \underline{H}) \cap bn(P) = \emptyset$. For rules (COM$_i$) and (PAR$_i$), consider $i \in \{0,1\}$, and let $\tilde{b} = \mathsf{dom}(\overline{H}'_i) \backslash \mathsf{dom}([\check{i}]\overline{H})$ and 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.

Define an independence relation on locations $I_l \subseteq \{0,1\}^* \times \{0,1\}^*$ *by*

$$(s_0, s_1) \in I_l \quad \textit{iff} \quad s_i = sis'_i,$$

where $i \in \{0,1\}$ *and* $s, s_0, s_1, s'_0, s'_1 \in \{0,1\}^*$. *Define the structural independence relation on events* $I_s \subseteq Ev \times Ev$ *by:*

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

As the following example shows, I_s misses the link dependencies.

Example 2.4. Consider the process $(\nu n)(\overline{a}\langle n\rangle \,||\, \underline{n}(x))$. According to I_s the two events $e = (\overline{a}\langle n\rangle, 0[\overline{a}\langle n\rangle][\mathbf{0}])$ and $e_m = (\underline{n}(m), 1[\underline{n}(x)][\mathbf{0}])$ are independent, for any $m \neq n$, but the semantics does not allow the input event to happen until after the name n has been extruded, i.e. there is an objective dependency between the extruding output and the input.

The next example illustrates that both names in an input action $\underline{m}(n)$ may have been previously extruded, giving rise to a conjunctive causality.

Example 2.5. Consider the process $P = (\nu n)(\nu m)(\overline{a}\langle n\rangle \,||\, (\overline{b}\langle m\rangle \,||\, \underline{m}(x)))$. From $(\emptyset, \emptyset) \vdash P$ we can have two extruding outputs on channels a and b of the names n respectively m, after which the output history would contain two pairs $\overline{H} = \{(n, 0[\overline{a}\langle n\rangle][\mathbf{0}]), (m, 10[\overline{b}\langle m\rangle][\mathbf{0}])\}$. We now may have an input action with label $\underline{m}(n)$ that depends on both extruders.

The final example shows that a name may have several parallel extruders, giving rise to a *disjunctive* causality.

Example 2.6. Consider the process $(\nu n)(\overline{a}\langle n\rangle \,||\, (\overline{b}\langle n\rangle \,||\, \underline{n}(x)))$, which has two *parallel extruders* $\overline{a}\langle n\rangle$ and $\overline{b}\langle n\rangle$. We have the three 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 $e_n = (\underline{n}(m), 11[\underline{n}(x)][\mathbf{0}])$. The event e_n for the input action is independent of both output events, but it cannot happen before at least one has happened.

The extrusion histories to the left of the turnstile helps us take into account the link causalites.

Definition 2.7. *A* history $H \subseteq \mathcal{H} = \mathcal{N} \times \mathcal{L}$ *is a relation between names and prefix locations, and an* extrusion history $(\overline{H}, \underline{H})$ *is a pair of histories, referred to as the* output history *and* input history *respectively. For a history* H *and* $i \in \{0,1\}$ *let* iH *denote the history* $\{(n, iu) | (n, u) \in H\}$. *Let* $[\check{i}]H = \{(n, u) \mid (n, iu) \in H\}$ *and* $[i]H = \{(n, iu) \mid (n, iu) \in H\}$ *and* $[\check{\epsilon}]H = H$. *Finally, let* $\mathsf{dom}(H) = \{n \mid (n, u) \in H\}$, *i.e. the set of names recorded in the history.*

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

Definition 2.8. *Define the* causal *early semantics as the 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. $\mathsf{dom}(D) = \mathsf{dom}(\overline{H}) \cap no(\alpha)$,

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

Definition 2.9. *Let the set of* events **Ev** *be defined by:*

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

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*

$$e'_0 I_s\, e'_1 \;\wedge\; \nexists n : D_i(n) = u_{1-i} \quad \textit{for } i \in \{0,1\}.$$

Returning to Ex. 2.6, the event $e_n = (\underline{n}(m), 11[\underline{n}(x)][\mathbf{0}])$ will be split in two events $\big(e_n, (n, 0[\overline{a}\langle n\rangle][\mathbf{0}])\big)$ and $\big(e_n, (n, 10[\overline{b}\langle n\rangle][\mathbf{0}])\big)$ corresponding to transitions between the same two states.

We now briefly explain the rules in Fig. 1.

The (IN) rule is the standard early input rule, substituting a received name n for the parameter m in P, yielding $P' = P[m := n]$, and enriched by recording the prefix location $u = [\underline{a}(m).P][P']$ 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, except the prefix location $u = [\overline{a}\langle n\rangle.P][P']$ is added to the transition.

The (OPEN) is the standard open rule, except the prefix location is recorded for the extruded name n in the output history and not in the label, as is custom for the standard pi-semantics. Avoiding name extrusions in the labels ensures unique labels for events in Sec. 3, and only one (COM) rule.

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 extract a possibly extruded name from the histories by the set $\tilde{b} = \mathsf{dom}(\overline{H}'_i) \backslash \mathsf{dom}([\check{i}]\overline{H})$ and not from the action label α. In the location, we record in which branch of the parallel composition the action happened by prefixing with $i \in \{0, 1\}$. The extruders recorded in the set $\overline{H}''$ in the rules $(\text{PAR})_i$ captures exactly the parallel extrusion illustrated in Ex. 2.6. Specifically, an output prefix 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. We illustrate the use of the input history in the example below.

Example 2.10. 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. $(\emptyset, \emptyset) \vdash P \xrightarrow[1[\underline{b}(x).\overline{c}\langle n\rangle][\overline{c}\langle n\rangle]]{\underline{b}(m)} (\emptyset, \{(m, 1)\}) \vdash (\nu n)(\overline{a}\langle n\rangle \,||\, \overline{c}\langle n\rangle)$, with $m \neq n$.

After the first transition we may both be receiving n or a name $m \neq n$:

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

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

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.

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. We combine the communication and close rules by abuse of notation, writing $(\nu\emptyset)P$ for P in the (COM_i) rules, thereby combining the standard (CLOSE) rule for communication of a bound name with that of communication of a free 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 forwards input histories from the sender to the receiver via the set $\underline{H}''$.

We end by stating the result that the standard interleaving, early operational semantics can be obtained from the rules in Fig. 1 by ignoring the locations and extract only the scope extrusion from the histories.

Proposition 2.11. *For a pi-process P, the transition system reachable from P using the standard interleaving, early operational semantics is bisimilar to the transition system $(\mathbf{Proc}_P, (\emptyset, \emptyset) \vdash P, \Lambda, \rightarrow_\pi)$, where*

- $\mathbf{Proc}_P = \{(\overline{H}, \underline{H}) \vdash P' \mid (\emptyset, \emptyset) \vdash P \rightarrow^* (\overline{H}, \underline{H}) \vdash P'\}$, *and $\rightarrow^*$ is the transitive closure of the transition relation in Fig. 1*
- *$\lambda \in \Lambda$ is defined by the grammar $\lambda ::= (\nu n)\overline{m}\langle n\rangle \mid \overline{m}\langle n\rangle \mid \underline{m}(n) \mid \tau$*
- *$(\overline{H}, \underline{H}) \vdash P \xrightarrow{\alpha'}_\pi (\overline{H}', \underline{H}') \vdash P'$ if $(\overline{H}, \underline{H}) \vdash P \xrightarrow[u]{\alpha} (\overline{H}', \underline{H}') \vdash P'$ for some $\overline{H}$, $\underline{H}$, $\overline{H}'$, $\underline{H}'$, u, and $\alpha' = (\nu n)\alpha$ if $\mathsf{dom}(\overline{H}') \backslash \mathsf{dom}(\overline{H}) = \{n\}$ and $\alpha' = \alpha$ otherwise.*

3 A stable non-interleaving early operational semantics

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[4] transition system (LATS) as defined in [1,23,26,12] and recalled in the following definition.

Definition 3.1. *A* labelled asynchronous transition system *(LATS) is a tuple* $(S, i, E, I, \mathcal{T}, lab, \mathcal{A})$ *such that*

- $(S, i, E, \mathcal{T})$ *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. $e_1 I e_2 \wedge \{(s, e_1, s_1), (s_1, e_2, s_3)\} \subseteq \mathcal{T} \;\Rightarrow\; \exists s_2 : \{(s, e_2, s_2), (s_2, e_1, s_3)\} \subseteq \mathcal{T}$*.*

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 [26, Ch.7].

Recalling the standard semantics derived in Prop. 2.11 we define the non-interleaving semantics as a labelled asynchronous transition system.

Definition 3.2. *The semantic rules for the pi-calculus that we gave in Fig. 1 generate a labelled asynchronous transition system* $TS_\pi(P) = (\mathbf{Proc}_P, (\emptyset, \emptyset) \vdash P, \mathbf{Ev}_P, I, \mathcal{T}, lab, \mathcal{A})$ *for a pi-process* P *where*

- $\big((\overline{H}, \underline{H}) \vdash P, e, (\overline{H}', \underline{H}') \vdash P'\big) \in \mathcal{T}$ *iff* $(\overline{H}, \underline{H}) \vdash P \xrightarrow[u,D]{\alpha} (\overline{H}', \underline{H}') \vdash P'$ *and* $e = \big((\alpha, u), D\big) \in \mathbf{Ev}$*,*
- $\mathbf{Ev}_P = \{e \mid (q, e, q') \in \mathcal{T}\}$
- $lab\big((\alpha, u), D\big) = \alpha$*,*
- $\alpha \in \mathcal{A}$ *is the set of labels generated by the grammar* $\alpha ::= \overline{a}\langle n\rangle \mid \underline{a}(n) \mid \tau$

Theorem 3.3. *The transition system given in Definition 3.2 is a labelled asynchronous transition system.*

That the semantics in 3.2 satisfy the first property of Def. 3.1 follows trivially from the definition. The following lemma states that the transition system is event deterministic, i.e. that it satisfies property 2. of Def. 3.1.

Lemma 3.4. *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* $P' = P''$*.*

To prove that the transition system given in Def. 3.2 satisfies the last two (diamond) properties of a labelled asynchronous transition system we need some intermediate results, following the approach in [18].

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

Definition 3.5 (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 3.6. *For any* $s, s' \in \{0,1\}^*$ *and any process* P*, whenever Comp is defined, we have* $Comp(s, Comp(s', P)) = Comp(s's, P)$*.*

From any transition we can recover the transition in the immediate component.

[4] asynchronous here refers to non-interleaving, not the style of communication.

Lemma 3.7. *For $s \in \{\varepsilon, 0, 1\}$ and $s' \in \{\varepsilon, 0, 1\}^*$ we have $(\overline{H}, \underline{H}) \vdash P \xrightarrow[ss'u_\varepsilon]{\alpha} (\overline{H}', \underline{H}') \vdash P'$ if*

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

where u_ε is either $[P''][P''']$ 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 $\alpha' = \alpha$ and $bn(\alpha) \in fn(Comp(1, P))$;*
2. *when $s = 1$ we have $\alpha' = \alpha$ and $bn(\alpha) \in fn(Comp(0, P))$;*
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, we either have ($\tilde{n} \cap n(\alpha) = \emptyset$ and $\alpha' = \alpha$) or ($\exists b \in \tilde{n} : \alpha = \overline{a}\langle b\rangle$ and $\alpha' = \overline{a}\langle b\rangle$ with $a \notin \tilde{n}$);*
4. *when $s = \varepsilon$ and $P = !P_1$ we have $\alpha' = \alpha$.*

Applying several times Lemma 3.7, 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 3.8. *For location strings $s, s_0, s_1, s'_0, s'_1 \in \{0, 1\}^*$ we have*

$$(\overline{H}, \underline{H}) \vdash P \xrightarrow[s\langle 0s_0s'_0[P_0][P'_0], 1s_1s'_1[P_1][P'_1]\rangle]{\tau} (\overline{H}', \underline{H}') \vdash P'$$

if

$$([\check{s}_l]\overline{H}, [\check{s}_l]\underline{H}) \vdash Comp(s0s_0, P) \xrightarrow[s'_0[P_0][P'_0]]{\alpha} (\overline{H}'', \underline{H}'') \vdash Comp(s0s_0, P')$$

and

$$([\check{s_r}]\overline{H}, [\check{s_r}]\underline{H}) \vdash Comp(s1s_1, P) \xrightarrow[s'_1[P_1][P'_1]]{\overline{\alpha}} (\overline{H}''', \underline{H}''') \vdash Comp(s1s_1, P')$$

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

Conversely, we can lift a transition from a component.

Lemma 3.9. *For $s \in \{\varepsilon, 0, 1\}$ we have that*

$$\text{if} \quad ([\check{s}]\overline{H}, [\check{s}]\underline{H}) \vdash Comp(s, P) \xrightarrow[s'[P_0][P'_0]]{\alpha} P'_1$$

then

$$(\overline{H}, \underline{H}) \vdash P \xrightarrow[ss'[P_0][P'_0]]{\alpha'} P' \text{ with } Comp(s, P') = P'_1$$

and α' defined in terms of α, under the following restrictions:

1. *for $s = 0$ if $bn(\alpha) \in fn(Comp(1, P))$ and $\alpha' = \alpha$;*
2. *for $s = 1$ if $bn(\alpha) \in fn(Comp(0, P))$ and $\alpha' = \alpha$;*
3. *for $s = \varepsilon$ and $P = (\nu n)P_1$ if ($n \in fn(\alpha)$ and $\alpha' = \alpha$) or ($\alpha = \overline{a}\langle n\rangle$ and $\alpha' = \overline{a}\langle n\rangle$ and $n \neq a$).*

Applying several times Lemma 3.9, when the needed restrictions exist, we can extend s to be a string of location components: $s \in \{0, 1\}^*$.

Lemma 3.10. *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 $a \notin \mathsf{dom}(\overline{H}_0 \backslash \overline{H}'_0)$, 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 $Comp(1, P') = P''_1$. A symmetric case was elided.

Lemma 3.11. *For any process P and a location string s then*

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

We end by noting that the non-interleaving semantics is a conservative extension of the one for CCS given in [18]. 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.

Proposition 3.12. *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 [18].*

4 Conclusion and Related work

We provided the first stable, non-interleaving operational semantics for the pi-calculus conservatively generalising the interleaving early operational semantics. The semantics is given as labelled asynchronous transition systems. We followed and conservatively generalised the approach for CCS in [18] by capturing the link causalities introduced in the pi-calculus processes by employing a notion of extrusion histories.

In the companion technical report [20] we have worked out the generalisation to unguarded choice, which makes the notion of location more complex, as also described in [6]. We are currently working on a more thorough comparison with the related work [17,13,5,3,21,7,9,11,10], in particular we aim to explore the differences betwen early and late style non-interleaving semantics. Finally, we work on extending this work to the psi-calculus [2].

References

1. M. A. Bednarczyk. *Categories of asynchronous systems*. PhD thesis, Univ. Sussex, 1988.
2. J. Bengtson, M. Johansson, J. Parrow, and B. Victor. Psi-calculi: a framework for mobile processes with nominal data and logic. *Logical Methods in Computer Science*, 7(1), 2011.
3. M. Boreale and D. Sangiorgi. A fully abstract semantics for causality in the pi-calculus. In *STACS*, pages 243–254, 1995.
4. G. Boudol, I. Castellani, M. Hennessy, and A. Kiehn. A theory of processes with localities. *Formal Asp. Comput.*, 6(2):165–200, 1994.
5. N. Busi and R. Gorrieri. A Petri Net Semantics for pi-Calculus. In *CONCUR*, volume 962 of *LNCS*, pages 145–159. Springer, 1995.
6. I. Castellani. Process algebras with localities. In *Handbook of Process Algebra*, chapter 15, pages 945–1045. Elsevier, 2001.
7. G. L. Cattani and P. Sewell. Models for Name-Passing Processes: Interleaving and Causal. In *LICS*, pages 322–333. IEEE Computer Society, 2000.
8. E. Clarke, O. Grumberg, M. Minea, and D. Peled. State space reduction using partial order techniques. *Int. Journal on Software Tools for Technology Transfer*, 2(3):279–287, 1999.
9. S. Crafa, D. Varacca, and N. Yoshida. Event structure semantics of parallel extrusion in the pi-calculus. In *FOSSACS*, volume 7213 of *LNCS*, pages 225–239. Springer, 2012.
10. I. Cristescu. *Operational and denotational semantics for the reversible π- calculus*. PhD thesis, Université Paris Diderot - Paris 7 - Sorbonne Paris Cité, 2015.
11. I. Cristescu, J. Krivine, and D. Varacca. A compositional semantics for the reversible pi-calculus. In *ACM/IEEE Symposium on Logic in Computer Science, LICS*, pages 388–397. IEEE Computer Society, 2013.
12. T. T. Hildebrandt. *Categorical Models for Concurrency: Independence, Fairness and Dataflow*. PhD thesis, University of Aarhus, Denmark, 1999.
13. L. J. Jagadeesan and R. Jagadeesan. Causality and True Concurrency: A Data-flow Analysis of the Pi-Calculus. In *AMAST*, volume 936 of *LNCS*, pages 277–291. Springer, 1995.
14. I. Lanese, C. A. Mezzina, and J.-B. Stefani. Reversibility in the higher-order pi-calculus. *Theoretical Computer Science*, 625:25–84, 2016.
15. R. Milner. *A Calculus of Communicating Systems*, volume 92. Springer-Verlag, 1980.
16. R. Milner, J. Parrow, and D. Walker. A Calculus of Mobile Processes, I-II. *Information and Computation*, 100(1):1–77, 1992.

17. U. Montanari and M. Pistore. Concurrent semantics for the pi-calculus. *Electr. Notes Theor. Comput. Sci.*, 1:411–429, 1995.
18. M. Mukund and M. Nielsen. CCS, Location and Asynchronous Transition Systems. In *FSTTCS*, volume 652 of *LNCS*, pages 328–341. Springer, 1992.
19. M. Nielsen, G. Plotkin, and G. Winskel. Petri nets, event structures and domains. In *Semantics of Concurrent Computation*, volume 70 of *LNCS*, pages 266–284. Springer, 1979.
20. H. Normann, C. Johansen, and 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`.
21. D. Sangiorgi. Locality and interleaving semantics in calculi for mobile processes. *Theor. Comput. Sci.*, 155(1):39–83, 1996.
22. D. Sangiorgi and D. Walker. *The π-Calculus: a Theory of Mobile Processes*. Cambridge Univ. Press, 2001.
23. M. W. Shields. Concurrent machines. *Computer Journal*, 28(5):449–465, 1985.
24. I. Ulidowski, I. Phillips, and S. Yuen. Concurrency and reversibility. In *Reversible Computation: 6th International Conference, RC 2014, Kyoto, Japan, July 10-11, 2014. Proceedings*, volume 8507 of *LNCS*, pages 1–14. Springer, 2014.
25. R. van Glabbeek and U. Goltz. Refinement of actions and equivalence notions for concurrent systems. *Acta Informatica*, 37(4/5):229–327, 2001.
26. G. Winskel and M. Nielsen. Models for concurrency. In S. Abramski, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, pages 1–148. Oxford, 1995.