Highlights

Event structures for the reversible early internal π-calculus

Eva Graversen, Iain Phillips, Nobuko Yoshida

- First early reversible π-calculus.
- Statically and dynamically reversible versions of the πI-calculus.
- Equivalence between the two πI-calculi.
- Denotational event structure semantics of statically reversible πI-calculus.
- Operational event structure semantics of dynamically reversible πI-calculus.
- Equivalence between the two event structure semantics.

Event structures for the reversible early internal π-calculus

Eva Graversen, Iain Phillips, Nobuko Yoshida

Imperial College London

Abstract

The π-calculus is a widely used process calculus, which models communications between processes and allows the passing of communication links. Various operational semantics of the π-calculus have been proposed, which can be classified according to whether transitions are unlabelled (so-called reductions) or labelled. With labelled transitions, we can distinguish early and late semantics. The early version allows a process to receive names it already knows from the environment, while the late semantics and reduction semantics do not. All existing reversible versions of the π-calculus use reduction or late semantics, despite the early semantics of the (forward-only) π-calculus being more widely used than the late. We introduce two reversible forms of the internal π-calculus; these are the first to use early semantics. The internal π-calculus is a subset of the π-calculus where every link sent by an output is private, yielding greater symmetry between inputs and outputs. One of the new reversible calculi uses static reversibility, where performing an action does not change the structure of the process, and the other uses dynamic reversibility, where performing an action moves it to a separate history. We show an operational correspondence between the two calculi. For the static calculus we define denotational event structure semantics, which generate an event structure inductively on the structure on the process. For the dynamic calculus we define operational event structure semantics, which generate an event structure based on a labelled asynchronous transition system. We describe a correspondence between the resulting event structures.

Keywords: Reversible Computations, π-calculus, Early Semantics, Event Structures, Static vs Dynamic Reversibility, Denotational vs Operational Semantics

1. Introduction

The π-calculus [22] is a widely used process calculus, which models communications between processes using input and output actions, and allows the passing of communication links. Various operational semantics of the π-calculus have been proposed, which can be classified according to whether transitions are unlabelled or labelled. Unlabelled transitions (so-called reductions) represent completed interactions. As observed in [29] they give us the internal behaviour of complete systems, whereas to reason compositionally about the behaviour of a system in terms of its components we need labelled transitions. With labelled transitions, we can distinguish early and late

semantics [23], with the difference being that early semantics allows a process to receive (free) names it already knows from the environment, while the late does not. This creates additional causation in the early case between those inputs and previous output actions making bound names free. All existing reversible versions of the π-calculus use reduction semantics [17, 30] or late semantics [8, 21]. However the early semantics of the (forward-only) π-calculus is more widely used than the late, partly because it has a sound correspondence with contextual congruences [24, 15].

We define πIH and πIK, the first reversible early π-calculi. The new calculi are reversible forms of the internal π-calculus, or πI-calculus [28], which is a subset of the π-calculus where every link sent by an output is bound (private), yielding greater symmetry between inputs and outputs. It has been shown that the asynchronous π-calculus can be encoded in the asynchronous form of the πI-calculus [2].

The π-calculus has two forms of causation. *Structural* causation, as one would find in CCS, comes directly from the structure of the process, e.g. in $a(b).c(d)$ the action $a(b)$ must happen before $c(d)$. *Link* causation, on the other hand, comes from one action making a name available for others to use, e.g. in the process $a(x)|\overline{b}(c)$, the event $a(c)$ will be caused by $\overline{b}(c)$ making c a free name. Note that link causation as in this example is present in the early form of the πI-calculus though not the late, since it is created by the process receiving one of its free names. Restricting ourselves to the πI-calculus, rather than the full π-calculus lets us focus on the link causation created by early semantics, since it removes the other forms of link causation present in the π-calculus.

We base πIH on the work of Hildebrandt *et al.* [14], which used extrusion histories and locations to define a stable non-interleaving early operational semantics for the π-calculus. Locations record which branch was taken in each parallel composition to get to a given subprocess, and what the contents of this subprocess were before and after an action was performed. Extrusion histories record which actions extruded and which actions received free names and at which locations. We extend the extrusion histories so that they contain enough information to reverse the πI-calculus, storing not only extrusions but also communications. As in [14], we use locations and extrusion histories to determine independence of actions. Locations tell us if actions took place in different parallel subprocesses, which means they are structurally independent, and extrusion histories record which actions at which locations made any names used in a subsequent action free. This definition of independence gives us a number of important properties as detailed in [19]; however it is different from most notions of independence of transitions in that we allow actions with conflicting causes to be independent. Despite this difference from the more common notion of independence considered in [19], we get all but one of the axioms and all the properties they describe for independence in reversible calculi.

Allowing processes to evolve, while moving past actions to a history separate from the process, is called dynamic reversibility, and was introduced in RCCS [10], before being applied to the π-calculus to create $\rho\pi$ [17]. By contrast, static reversibility, as introduced in CCSK [25], lets processes keep their structure during the computation, and annotations are used to keep track of the current state and how actions may be reversed. Our other calculus, πIK, is inspired by CCSK and the statically reversible π-calculus of [21], which use communication keys to denote past actions. To keep track

of link causation, keys are used in a number of different ways in [21]. In our case we can handle link causation by using keys purely to annotate the action which was performed using the key, and any names which were substituted during that action. To use keys in a reversible full π-calculus, it is necessary in [21] to annotate opened restrictions with the key of the action that opened the restriction, since this is a source of link causation. By restricting ourselves to the πI-calculus, we avoid this.

Although our two reversible variants of the πI-calculus have very different syntax and originate from different ideas, we show an operational correspondence between them in Theorem 4.8. We do this despite the extrusion histories containing more information than the keys, since they remember what bound names were before being substituted. The mapping from πIH to πIK bears some resemblance to the one presented from RCCS to CCSK in [16], though with some important differences. Crucially, πIH uses centralised extrusion histories more similar to rhoπ [18] while RCCS uses distributed memories. Additionally, unlike CCS, πI has substitution as part of its transitions and memories are handled differently by πIK and πIH, and our mapping has to take this into account.

Event structures are a model of concurrency which describe causation, conflict and concurrency between events. They are 'truly concurrent' in that they do not reduce concurrency of events to the different possible interleavings. They have been used to model forward-only process calculi [7, 4, 31], including the πI-calculus [6]. Describing reversible processes as event structures is useful because it gives us a simple representation of the causal relationships between actions and gives us equivalences between processes which generate isomorphic event structures. True concurrency in semantics is particularly important in reversible process calculi, as the order actions can reverse in depends on their causal relations [26].

Event structure semantics of dynamically reversible process calculi have the added complexity of the histories and the actions in the process being separated, obscuring the structural causation. This was an issue for Cristescu *et al.* [9], who used rigid families [5], related to event structures, to describe the semantics of Rπ [8]. Their semantics require a process to first reverse all actions to find the original process, map this process to a rigid family, and then apply each of the reversed memories in order to reach the current state of the process. Aubert and Cristescu [1] used a similar approach to describe the semantics of a subset of RCCS processes as configuration structures.

We describe denotational structural event structure semantics of πIK, partly inspired by [7, 6], using a reversible form of bundle event structures [20], first introduced in [13]. Reversible event structures [27] allow their events to reverse and include relations describing when events can reverse. Bundle event structures are more expressive than prime event structures, since they allow an event to have multiple possible conflicting causes. This lets us model parallel composition without having a single action correspond to multiple events. While it would be possible to model πIK using reversible prime event structures, using bundle event structures not only gives us fewer events, it also lays the foundation for adding rollback to πIK and πIH, similarly to [13], which cannot be done using reversible prime event structures. We can use the same semantics for choice and output in the πI-calculus as in CCS [13], but due to using early semantics, a πI-calculus input needs to create an event for every possible name the action can receive. Additionally, in the denotational semantics of parallel composition, these

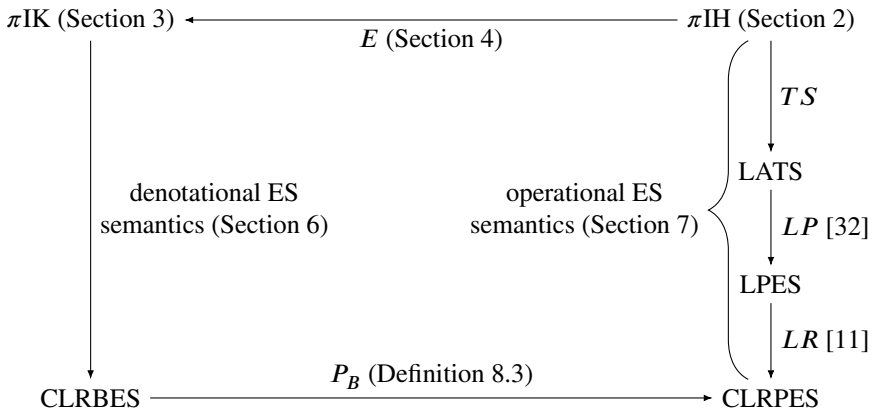

Figure 1: The reversible πI-calculi and event structures and mappings between them. The mapping TS generates a labelled transition system, LP maps from there to a prime event structure, which LR maps to a causal reversible prime event structure. The mapping P_B is a functor from reversible bundle to reversible prime event structures.

input events will get link causation added if the other part of the process has an output of the name being received. For these reasons, if we were to map a CCSK process to a πIK process, they would have different event structure semantics, and interestingly both differences are caused by the semantics being early.

Our event structure semantics of πIH are operationally defined and derived from the labelled asynchronous transition system (LATS) semantics of [14], which we can turn into labelled reversible prime event structures [27], using mappings from [32] and [11]. This means each event is a set of possible traces in the process, all of which end with the same action and vary only in the order in which independent actions are performed and whether they include any actions which are independent of the final action. We are able to generate these event structures despite our independence relation allowing actions with conflicting causes to be independent because the requirements for independent action in an LATS only discuss coinitial or sequential actions. Since the semantics of our process calculi are causally reversible, so are the event structures they generate. The mappings between our calculi and event structures are shown in Figure 1.

In addition to an operational correspondence between the two versions of reversible πI-calculus, we show two results about the correspondence between the event structure semantics. For any process with no past actions, we have two inverse morphisms between the event structure generated by the denotational semantics and a subset of the events in the event structure generated by the operational semantics (Theorem 8.7). The morphism from the subset of the events in the event structure generated by the operational semantics to the event structure generated by the denotational semantics can also be expanded, if one ignores labels, to map all the events of the event structure generated by the operational semantics to events from event structure generated by the denotational semantics (Theorem 8.11). This is illustrated by Figure 2.

The reason the operational semantics generate events with labels which do not exist in the denotational semantics is that the operational semantics include actions re-

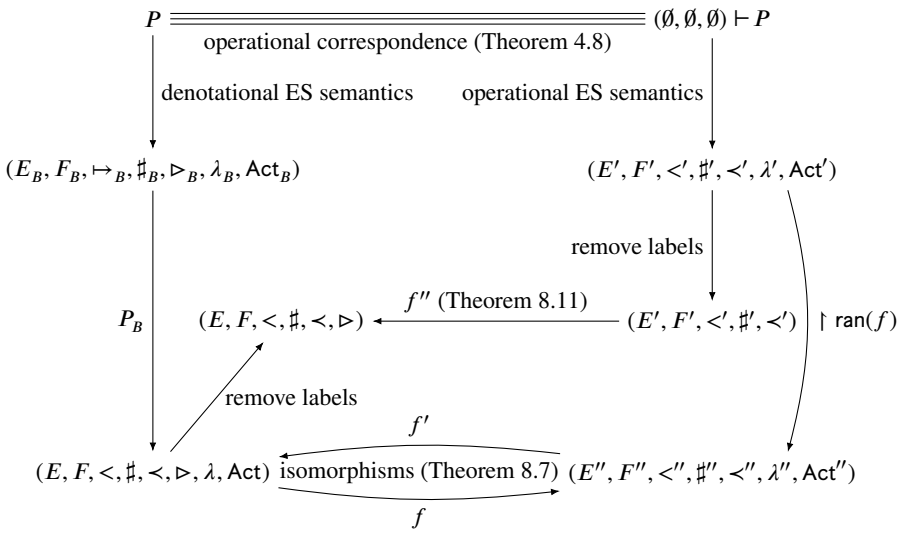

Figure 2: A reversible process, P, with no past actions, expressed in the two calculi and the relationship between them and the event structures generated by applying the two semantics to P. The functor P_B is a mapping from reversible bundle to reversible prime event structures. we have three mophisms f, f', and f'' such that $f' = f'' \restriction \mathsf{ran}(f)$.

quiring α-conversion, whereas the denotational semantics assume that the process has already been α-converted so that all free and bound names are different, and only use α-conversion when expanding a replication. This means that in a simple process $\overline{b}(a)$, the denotational semantics will generate one event, labelled $\overline{b}(a)$, while the operational semantics will generate an infinite set of events, with the labels $\{\overline{b}(x) \mid x \text{ is a name}\}$, and the inverse morphisms discussed above will be between the events labelled $\overline{b}(a)$ in the two event structures, while the expanded morphism ignoring labels will map every event in the second event structure to $\overline{b}(a)$.

The structure of the paper is as follows: Section 2 describes πIH; Section 3 describes πIK; Section 4 describes the mapping from πIH to πIK; Section 5 recalls labelled reversible bundle event structures from [13]; and Section 6 gives event structure semantics of πIK; Section 7 recalls mappings from [32, 11, 14] and uses them to get operational event structure semantics of πIH; and Section 8 shows a correspondence between the two event structure semantics.

Changes from [12]:.

- Sections 7 and 8 are entirely new, as the conference version focussed on using πIK as an intermediary in getting denotational event structure semantics of πIH. The introduction and conclusion have been modified to reflect the change in focus to comparing operational and denotational event structure semantics.
- Figures 1 and 2 are new as they include references to Sections 7 and 8.

- The causal operational semantics of πIH, which are necessary for operational event structure semantics, have been added to Section 2.
- Definitions 2.16 and 2.19, Lemmas 2.17 and 2.18, and Propositions 2.20 and 2.21 based on[19] are new.
- A mapping from πIH to the full π-calculus defined by [14] is described in Definition 2.23.
- The following results regarding πIK have been added: Propositions: 3.6, 3.7, and 3.13, Theorem 3.9, and Lemma 3.8.
- We use the mapping from πIH to πIK to transfer a property of πIK to πIH in Proposition 4.9.
- Theorems 6.10 and 6.11 proving correspondence between reverse transitions of the operational and denotational semantics of πIK are added.
- Proofs of results are included, as well as intermediate results Lemmas 2.5, 3.3, 3.4, 4.4, 6.6, and Propositions 6.3, 6.4, 6.5

2. πI-calculus reversible semantics with extrusion histories

Stable non-interleaving, early operational semantics of the π-calculus were defined in [14], using locations and extrusion histories to keep track of link causation. We will in this section use a similar approach to define a reversible variant of the πI-calculus, πIH, using the locations and histories to keep track of not just causation, but also past actions. The πI-calculus is a restricted variant of the π-calculus wherein output binds the name being sent, so in $\overline{a}(b).P$ name b is bound, corresponding to the π-calculus process $(\nu b)\overline{a}\langle b\rangle.P$. The syntax of πIH processes is:

$P ::= \sum_{i\in I} \alpha_i.P_i \mid P_0|P_1 \mid (\nu x)P \mid !P \quad \alpha ::= \overline{a}(b) \mid a(b)$

The forward semantics of πIH can be seen in Table 1 and the reverse semantics in Table 2. We associate each transition with an action $\mu ::= \alpha \mid \tau$ and a location u (Definition 2.1), describing where the action came from and what changes are made to the process as a result of the action. We store these location and action pairs in extrusion and communication histories associated with processes, so $(\overline{H}, \underline{H}, H) \vdash P$ means that if (μ, u) is an action and location pair in the output history $\overline{H}$ then μ is an output action, which P previously performed at location u. Similarly $\underline{H}$ contains pairs of input actions and locations and H contains triples of two communicating actions and the location associated with their communication. We use **H** as shorthand for $(\overline{H}, \underline{H}, H)$.

Definition 2.1 (Location [14]). A location u of an action μ is of one of the following forms:

1. $l[P][P']$ if μ is an input or output, where $l \in \{0,1\}^*$ describes the path taken through parallel compositions to get to μ's origin, P is the subprocess reached by following the path before μ has been performed, and P' is the result of performing μ in P.

2. $l\left\langle 0l_0[P_0][P'_0], 1l_1[P_1][P'_1]\right\rangle$ if $\mu = \tau$, where $l0l_0[P_0][P'_0]$ and $l1l_1[P_1][P'_1]$ are the locations of the two actions communicating.

Location l may be empty if the action was performed at the outer level of the process before any parallel composition.

We also use the operations on extrusion histories from Definition 2.2. These (1) add a branch to the path in every location, (2) isolate the extrusions whose locations begin with a specific branch, (3) isolate the extrusions whose locations begin with a specific branch and then remove the first branch from the locations, and (4) add a pair to the history it belongs in.

Definition 2.2 (Operations on extrusion histories [14]). Given an extrusion history $(\overline{H}, \underline{H}, H)$, for $H^* \in \{\overline{H}, \underline{H}, H\}$ we have the following operations for $i \in \{0, 1\}$:

1. $iH^* = \{(\mu, iu) \mid (\mu, u) \in H^*\}$
2. $[i]H^* = \{(\mu, iu) \mid (\mu, iu) \in H^*\}$
3. $[\check{i}]H^* = \{(\mu, u) \mid (\mu, iu) \in H^*\}$
4. $\mathbf{H} + (\mu, u) = \begin{cases} (\overline{H} \cup \{(\mu, u)\}, \underline{H}, H) & \text{if } (\mu, u) = (\overline{a}(n), u) \\ (\overline{H}, \underline{H} \cup \{(\mu, u)\}, H) & \text{if } (\mu, u) = (a(x), u) \\ (\overline{H}, \underline{H}, H \cup \{(\mu, u)\}) & \text{if } (\mu, u) = ((a(x), \overline{a}(n)), l\langle u_0, u_1\rangle) \end{cases}$

The forwards semantics of πIH have six rules. In [OUT] the action is an output, the location is the process before and after doing the output, and they are added to the output history. The equivalent reverse rule, [OUT^{-1}], similarly removes the pair from the history and transforms the process from the second part of the location back to the first. The input rule [IN] works similarly, but performs a substitution on the received name and adds the pair to the input history instead. In [PAR$_i$] we isolate the parts of the histories whose locations start with i and use those to perform an action in P_i, getting $\mathbf{H}'_i \vdash P'_i$. It then replaces the part of the histories parts of the histories whose locations start with i with $\mathbf{H}'_i$ when propagating the action through the parallel. A communication in [COM$_i$] adds memory of the communication to the history. The rules [SCOPE] and [STR] are standard and self-explanatory.

The reverse rules use the extrusion histories to find a location $l[P][P']$ such that the current state of the subprocess at l is P', and change it to P.

In these semantics we define structural congruence as consisting of α-conversion, $(\nu\, a)(\nu b)P \equiv (\nu\, b)(\nu\, a)P$, and $!P \equiv !P|P$. Here structural congruence is primarily used to create and remove extra copies of a replicated process when reversing the action that happened before the replication. Since we use locations in our extrusion histories, we try to avoid using structural congruence any more than necessary. However, not using it for parallel composition would mean that we would need some other way of preventing traces such as $\mathbf{H} \vdash !P \xrightarrow[u]{\mu} \overset{\mu}{\underset{u}{\rightsquigarrow}} \mathbf{H} \vdash !P|P$, which allows a process to reach a state it could not reach via a parabolic trace. Using structural congruence for replication does not cause any problems for the locations, as we can tell past actions originating in each copy of P apart by the path in their location, with actions from the ith copy having a path of i 0s followed by a 1. We show in Example 2.3 how actions and locations are stored in the histories.

$$\frac{u = [\sum_{i\in I} \alpha_i.P_i][P'_j] \quad \alpha_j = \overline{a}(n) \quad j \in I \quad (\overline{b}(n), u') \notin \overline{H}}{\mathbf{H} \vdash \sum_{i\in I} \alpha_i.P_i \xrightarrow[u]{\alpha_j} (\overline{H} \cup \{(\overline{a}(n), u)\}, \underline{H}, H) \vdash P_j} \text{ [OUT]}$$

$$\frac{u = [\sum_{i\in I} \alpha_i.P_i][P_j] \quad P'_j = P_j[x := n] \quad \alpha_j = a(x) \quad j \in I}{\mathbf{H} \vdash \sum_{i\in I} \alpha_i.P_i \xrightarrow[u]{a(n)} (\overline{H}, \underline{H} \cup \{(a(n), u)\}, H) \vdash P'_j} \text{ [IN]}$$

$$\frac{\begin{array}{c}([\check{i}]\overline{H}, [\check{i}]\underline{H}, [\check{i}]H) \vdash P_i \xrightarrow[u]{\mu} \mathbf{H}'_i \vdash P'_i \quad P'_{1-i} = P_{1-i} \\ \text{if } \mu = \overline{a}(n) \text{ then } n \notin \mathsf{fn}(P_{1-i}) \text{ and } (\overline{b}(n), u') \notin \overline{H}\end{array}}{\mathbf{H} \vdash P_0|P_1 \xrightarrow[iu]{\mu} ((\overline{H} \setminus [i]\overline{H}) \cup i\overline{H'_i}, (\underline{H} \setminus [i]\underline{H}) \cup i\underline{H'_i}, (H \setminus [i]H) \cup iH'_i) \vdash P'_0|P'_1} \text{ [PAR}_i\text{]}$$

$$\frac{\begin{array}{c}([\check{i}]\overline{H}, [\check{i}]\underline{H}, [\check{i}]H) \vdash P_i \xrightarrow[v_i]{\alpha_i} \mathbf{H}'_i \vdash P'_i \quad \alpha_i = \overline{a}(n) \quad \alpha_j = a(n) \\ ([\check{j}]\overline{H}, [\check{j}]\underline{H}, [\check{j}]H) \vdash P_j \xrightarrow[v_j]{\alpha_j} \mathbf{H}'_j \vdash P'_j \quad j = 1 - i \quad n \notin \mathsf{fn}(P_j)\end{array}}{\mathbf{H} \vdash P_0|P_1 \xrightarrow[(0v_0,1v_1)]{\tau} (\overline{H}, \underline{H}, H \cup \{((\alpha_0, \alpha_1), \langle 0v_0, 1v_1\rangle)\}) \vdash (\nu n)(P'_0|P'_1)} \text{ [COM}_i\text{]}$$

$$\frac{\mathbf{H} \vdash P \xrightarrow[u]{\mu} \mathbf{H}' \vdash P' \quad x \notin n(\mu)}{\mathbf{H} \vdash (\nu x)P \xrightarrow[u]{\mu} \mathbf{H}' \vdash (\nu x)P'} \text{ [SCOPE]} \qquad \frac{P \equiv P' \quad \mathbf{H} \vdash P' \xrightarrow[u]{\mu} \mathbf{H}' \vdash Q' \quad Q' \equiv Q}{\mathbf{H} \vdash P \xrightarrow[u]{\mu} \mathbf{H}' \vdash Q} \text{ [STR]}$$

Table 1: Semantics of πIH (forwards rules)

Example 2.3. *Consider the process $a(x).(\overline{x}(b)|x(y))|\overline{a}(c)$. If we start with empty histories, each transition adds actions and locations:*

$(\emptyset, \emptyset, \emptyset) \vdash a(x).(\overline{x}(b)|x(y))|\overline{a}(c)$

$\xrightarrow[1[\overline{a}(c)][0]]{\overline{a}(c)}$

$(\{(\overline{a}(c), 1[\overline{a}(c)][0])\}, \emptyset, \emptyset) \vdash a(x).(\overline{x}(b)|x(y))|0$

$\xrightarrow[0[a(x).(\overline{x}(b)|x(y))][(\overline{c}(b)|c(y))]]{a(c)}$

$(\{(\overline{a}(c), 1[\overline{a}(c)][0])\}, \{(a(c), 0[a(x).(\overline{x}(b)|x(y))][(\overline{c}(b)|c(y))])\}, \emptyset) \vdash (\overline{c}(b)|c(y))|0$

$\xrightarrow[0\langle 0[\overline{c}(b)][0], 1[c(y)][0]\rangle]{\tau}$

$(\{(\overline{a}(c), 1[\overline{a}(c)][0])\}, \{(a(c), 0[a(x).(\overline{x}(b)|x(y))][(\overline{c}(b)|c(y))])\},$
$\{((\overline{c}(b), c(b)), 0\left\langle 0[\overline{c}(b)][0], 1[c(y)][0]\right\rangle)\}) \vdash (\nu b)(0|0)|0$

The first transition adds an action and location to the output history, which allows the input to receive the same name, c. The input transition is then added to the input history, and finally the synchronisation history stores both of the actions involved in the final transition and their locations.

$$\frac{u = [\sum_{i\in I} \alpha_i.P_i][P_j] \quad \alpha_j = \overline{a}(n) \quad j \in I \quad (\overline{a}(n), u) \in \overline{H}}{\mathbf{H} \vdash P_j \overset{\alpha_j}{\underset{u}{\rightsquigarrow}} (\overline{H} \setminus \{(\overline{a}(n), u)\}, \underline{H}, H) \vdash \sum_{i\in I} \alpha_i.P_i} \ [\mathrm{OUT}^{-1}]$$

$$\frac{u = [\sum_{i\in I} \alpha_i.P_i][P'_j] \quad P'_j = P_j[x := n] \quad \alpha_j = a(x) \quad j \in I \quad (a(n), u) \in \underline{H}}{\mathbf{H} \vdash P'_j \overset{a(n)}{\underset{u}{\rightsquigarrow}} (\overline{H}, \underline{H} \setminus \{(a(n), u)\}, H) \vdash \sum_{i\in I} \alpha_i.P_i} \ [\mathrm{IN}^{-1}]$$

$$\frac{\begin{array}{l}([\check{i}]\overline{H}, [\check{i}]\underline{H}, [\check{i}]H) \vdash P_i \overset{\mu}{\underset{u}{\rightsquigarrow}} \mathbf{H}'_i \vdash P'_i \quad P'_{1-i} = P_{1-i} \\ \text{if } \mu = \overline{a}(n) \text{ then } n \notin \mathsf{fn}(P_{1-i})\end{array}}{\mathbf{H} \vdash P_0|P_1 \overset{\mu}{\underset{iu}{\rightsquigarrow}} ((\overline{H} \setminus [i]\overline{H}) \cup i\overline{H'_i}, (\underline{H} \setminus [i]\underline{H}) \cup i\underline{H'_i}, (H \setminus [i]H) \cup iH'_i) \vdash P'_0|P'_1} \ [\mathrm{PAR}_i^{-1}]$$

$$\frac{\begin{array}{l}([\check{i}]\overline{H} \cup \{(\overline{a}(n), v_i)\}, [\check{i}]\underline{H}, [\check{i}]H) \vdash P_i \overset{\overline{a}(n)}{\underset{v_i}{\rightsquigarrow}} \mathbf{H}'_i \vdash P'_i \\ ([\check{j}]\overline{H}, [\check{j}]\underline{H} \cup \{(a(n), v_j)\}, [\check{j}]H) \vdash P_j \overset{a(n)}{\underset{v_j}{\rightsquigarrow}} \mathbf{H}'_j \vdash P'_j \\ j = 1 - i \quad n \notin \mathsf{fn}(P_j) \quad ((a(n), \overline{a}(n)), \langle 0v_0, 1v_1\rangle) \in H\end{array}}{\begin{array}{l}\mathbf{H} \vdash (\nu n)(P_0|P_1) \overset{\tau}{\underset{\langle 0v_0, 1v_1\rangle}{\rightsquigarrow}} \\ \quad (\overline{H}, \underline{H}, H \setminus \{((a(n), \overline{a}(n)), \langle 0v_0, 1v_1\rangle)\}) \vdash P'_0|P'_1\end{array}} \ [\mathrm{COM}_i^{-1}]$$

$$\frac{\mathbf{H} \vdash P \overset{\mu}{\underset{u}{\rightsquigarrow}} \mathbf{H}' \vdash P' \quad x \notin n(\alpha)}{\mathbf{H} \vdash (\nu x)P \overset{\mu}{\underset{u}{\rightsquigarrow}} \mathbf{H}' \vdash (\nu x)P'} \ [\mathrm{SCOPE}^{-1}] \qquad \frac{P \equiv P' \quad \mathbf{H} \vdash P' \overset{\mu}{\underset{u}{\rightsquigarrow}} \mathbf{H}' \vdash Q' \quad Q' \equiv Q}{\mathbf{H} \vdash P \overset{\mu}{\underset{u}{\rightsquigarrow}} \mathbf{H}' \vdash Q} \ [\mathrm{STR}^{-1}]$$

Table 2: Semantics of reversible πIH-calculus (reverse rules)

We will show that our calculus reverses in a causal manner from any reachable process (Definition 2.4). We use $\rightarrowtail$ to denote that an action may be forwards or reverse.

Definition 2.4 (Reachable Process). Let P be a process. We say that $\mathbf{H} \vdash P$ is reachable, denoted $\mathsf{reach}(\mathbf{H} \vdash P)$, if there exists a process $(\emptyset, \emptyset, \emptyset) \vdash Q$ such that $(\emptyset, \emptyset, \emptyset) \vdash Q \rightarrowtail^* \mathbf{H} \vdash P$.

We say that $\mathbf{H} \vdash P$ is forwards reachable if there exists a standard process $(\emptyset, \emptyset, \emptyset) \vdash$ such that $(\emptyset, \emptyset, \emptyset) \vdash \rightarrow^* \mathbf{H} \vdash P$.

We then show a correspondence between forward and reverse transitions.

Lemma 2.5. *Let P be a process. If there exists an extrusion history $\mathbf{H}$ such that $\mathbf{H} \vdash P \overset{\mu}{\underset{u}{\rightsquigarrow}} \mathbf{H}' \vdash P'$ then there exists (μ', u') such that $\mathbf{H} = \mathbf{H}' + (\mu', u')$, and for any extrusion history $\mathbf{H}''$ not containing (μ', u'), $\mathbf{H}'' + (\mu', u') \vdash P \overset{\mu}{\underset{u}{\rightsquigarrow}} \mathbf{H}'' \vdash P'$.*

PROOF. [SCOPE^{-1}] and [PAR$_i^{-1}$] simply propagate the changes to extrusion histories, and [COM$_i^{-1}$], [IN^{-1}], and [OUT^{-1}] remove exactly one extrusion from the histories, which is the only one they depend on.

Proposition 2.6 (Loop).

1. *Given a forwards reachable πIH-calculus process P and an extrusion history* $\mathbf{H}$*, if* $\mathbf{H} \vdash P \xrightarrow[u]{\alpha} \mathbf{H}' \vdash Q$*, then* $\mathbf{H}' \vdash Q \overset{\alpha}{\underset{u}{\rightsquigarrow}} \mathbf{H} \vdash P$.
2. *Given a forwards-reachable πIH-calculus process P and an extrusion history* $\mathbf{H}$*, if* $\mathbf{H} \vdash P \overset{\alpha}{\underset{u}{\rightsquigarrow}} \mathbf{H}' \vdash Q$*, then* $\mathbf{H}' \vdash Q \xrightarrow[u]{\alpha} \mathbf{H} \vdash P$.

PROOF. We prove this by induction on the transitions in Appendix B.1.

To get causal semantics of πIH, we add a set of causes, D, to each transition. Set D associates the non-output names of an action with the locations where they were extruded, and thereby tells us which actions extruded those names necessary for the current action. If the non-output names in an action were always free then D is empty.

Definition 2.7 (Causal semantics [14]). The early causal semantics consist of transitions of the form $\mathbf{H} \vdash P \xrightarrow[u,D]{\alpha} \mathbf{H}' \vdash P'$ where $\mathbf{H} \vdash P \xrightarrow[u]{\alpha} \mathbf{H}' \vdash P'$ and

1. $(n, u) \in D \Rightarrow \exists a.\ (\overline{a}(n), u) \in \overline{H}$;
2. if $(n, u), (n, u') \in D$ then $u = u'$;
3. $(n, u) \in D$ if and only if $(\overline{a}(n), u') \in \overline{H}$ for some a, u' and $n \in \mathsf{no}(\alpha)$ where $\mathsf{no}(\alpha)$ is the set of non-output names in α, defined by $\mathsf{no}(\overline{a}(b)) = \{a\} \setminus \{b\}$, $\mathsf{no}(a(b)) = \{a, b\}$ and $\mathsf{no}(\tau) = \emptyset$.

This lets us define a notion of independence on locations and causes. Our definition of independence differs from the notions of concurrency used by other reversible process calculi, in that it is not just defined on coinitial [10] or composable [8] transitions. Having the definition apply to any two transitions becomes important in our event structure semantics in Section 7.

Definition 2.8 (Path). Given a location u, its set of paths is defined as

$$\mathsf{loc}(u) = \begin{cases} \{l\} & \text{if } u = l[P][P'] \\ \{ll_0, ll_1\} & \text{if } u = l\left\langle l_0[P_0][P_0'], l_1[P_1][P_0]\right\rangle \end{cases}$$

Definition 2.9 (Independence [14]). Two locations, u_0 and u_1, are independent, denoted $u_0 I u_1$, if for all $l_0 \in \mathsf{loc}(u_0)$ and $l_1 \in \mathsf{loc}(u_1)$, there exist l, l_0', l_1' such that either $l_0 = l0l_0'$ and $l_1 = l1l_1'$ or $l_0 = l1l_0'$ and $l_1 = l0l_1'$.

Two transitions $\overset{\alpha_0}{\underset{u_0,D_0}{\rightarrowtail}}$ and $\overset{\alpha_1}{\underset{u_1,D_1}{\rightarrowtail}}$ are independent if u_0 and u_1 are independent, and there does not exist n such that $D_i(n) = u_{1-i}$.

We prove the square property using our definition of independence of transitions. Unlike Rπ [8], we are not keeping track of which restriction was opened by each output action. We therefore get transitions to P' with the same labels as the original, rather than equivalent labels.

Proposition 2.10 (Square [10]). *If* $\mathbf{H} \vdash P \xrightarrow[u_0, D_0]{\mu_0} \mathbf{H}_0 \vdash P_0$ *and* $\mathbf{H} \vdash P \xrightarrow[u_1, D_1]{\mu_1} \mathbf{H}_1 \vdash P_1$ *are independent transitions then there exists* $\mathbf{H}' \vdash P'$ *such that* $\mathbf{H}_0 \vdash P_0 \xrightarrow[u_1, D_1]{\mu_1} \mathbf{H}' \vdash P'$ *and* $\mathbf{H}_1 \vdash P_1 \xrightarrow[u_0, D_0]{\mu_0} \mathbf{H}' \vdash P'$.

PROOF. See Appendix B.2.

Unlike in the previous section, since we have an explicit notion of independence, we can define trace equivalence. We can then use trace equivalence to get an alternative description of the parabolic property we want our reversible calculus to have. We use t to refer to traces of the form $\mathbf{H}_0 \vdash P_0 \xrightarrow[u_0, D_0]{\mu_0} \mathbf{H}_1 \vdash P_1 \dots \xrightarrow[u_{n-1}, D_{n-1}]{\mu_{n-1}} \mathbf{H}_n \vdash P_n$, and $\underline{\mathsf{t}}$ to refer to the reversal of t, which according to Proposition 2.6 is a trace if and only if t is. We use $\mathsf{t}; \mathsf{t}'$ to compose traces, and ε to refer to the empty trace.

Definition 2.11 (Causal equivalence [10]). We define trace equivalence $\sim$ as the least equivalence relation closed under composition such that:

$$\mathsf{t}; \underline{\mathsf{t}} \sim \varepsilon$$
$$\underline{\mathsf{t}}; \mathsf{t} \sim \varepsilon$$
$$\mathbf{H} \vdash P \xrightarrow[u_0, D_0]{\mu_0} \xrightarrow[u_1, D_1]{\mu_1} \mathbf{H}' \vdash P' \sim \mathbf{H} \vdash P \xrightarrow[u_1, D_1]{\mu_1} \xrightarrow[u_0, D_0]{\mu_0} \mathbf{H}' \vdash P' \text{ if } \xrightarrow[u_1, D_1]{\mu_1} \text{ and } \xrightarrow[u_0, D_0]{\mu_0} \text{ are independent}$$

Lemma 2.12. *Given a forwards-reachable process* $\mathbf{H} \vdash P$ *with possible transitions* $\mathbf{H} \vdash P \overset{\alpha_0}{\rightsquigarrow}_{u_0, D_0} \mathbf{H}_0 \vdash P_0$ *and* $\mathbf{H} \vdash P \overset{\alpha_1}{\rightsquigarrow}_{u_1, D_1} \mathbf{H}_1 \vdash P_1$, *either* $\alpha_0 = \alpha_1$, $u_0 = u_1$, $D_0 = D_1$, *and* $\mathbf{H}_0 \vdash P_0 \equiv \mathbf{H}_1 \vdash P_1$, *or* $\overset{\alpha_0}{\rightsquigarrow}_{u_0, D_0}$ *and* $\overset{\alpha_1}{\rightsquigarrow}_{u_1, D_1}$ *are independent.*

PROOF. See Appendix B.3.

Lemma 2.13. *Given a process,* $\mathbf{H} \vdash P$, *we cannot have an infinite reverse trace* $\mathbf{H} \vdash P \twoheadleftarrow^*$.

PROOF. Each reverse transition removes an element from $\mathbf{H}$. Since $\mathbf{H}$ is finite, so is every reverse trace.

Theorem 2.14 (Parabola [10]). *Let* t *be a trace, then there exists a forward trace* t_f *and a backward trace* t_b *such that* $\mathsf{t} \sim \mathsf{t}_b; \mathsf{t}_f$.

PROOF. Follows from Proposition 2.10 and Lemma 2.12 according to [19].

Theorem 2.15 (Trace equivalence [10]). *Let* t *and* t' *be cofinal and coinitial traces. Then* $\mathsf{t} \sim \mathsf{t}'$.

PROOF. Follows from Theorem 2.14 and Lemma 2.13 according to [19].

Theorems 2.14 and 2.15 are not the only results from [19] we can apply to the πIH calculus. We gain several interesting properties by proving some simple results, both of which follow easily from our definition of independence. First we recall their definition of an event.

Definition 2.16 (Events [19]). Let $\sim$ be the smallest equivalence relation on transitions satisfying: if $\mathbf{H}_P \vdash P \overset{\alpha_0}{\underset{u_0,D_0}{\rightarrowtail}} \mathbf{H}_Q \vdash Q$, $\mathbf{H}_P \vdash P \overset{\alpha_1}{\underset{u_1,D_1}{\rightarrowtail}} \mathbf{H}_R \vdash R$, $\mathbf{H}_Q \vdash Q \overset{\alpha_1}{\underset{u_1,D_1}{\rightarrowtail}} \mathbf{H}_S \vdash S$, $\mathbf{H}_R \vdash R \overset{\alpha_0}{\underset{u_0,D_0}{\rightarrowtail}} \mathbf{H}_S \vdash S$, $\overset{\alpha_0}{\underset{u_0,D_0}{\rightarrowtail}}$ and $\overset{\alpha_1}{\underset{u_1,D_1}{\rightarrowtail}}$ are independent, and

- $\mathbf{H}_Q \vdash Q \neq \mathbf{H}_R \vdash R$ if $\overset{\alpha_0}{\underset{u_0,D_0}{\rightarrowtail}}$ and $\overset{\alpha_1}{\underset{u_1,D_1}{\rightarrowtail}}$ are either both forwards or both backwards;
- $\mathbf{H}_P \vdash P \neq \mathbf{H}_S \vdash S$ otherwise;

then $\mathbf{H}_P \vdash P \overset{\alpha_0}{\underset{u_0,D_0}{\rightarrowtail}} \mathbf{H}_Q \vdash Q \sim \mathbf{H}_R \vdash R \overset{\alpha_0}{\underset{u_0,D_0}{\rightarrowtail}} \mathbf{H}_S \vdash S$. The equivalence classes of forward transitions, written $[\mathbf{H}_P \vdash P \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_Q \vdash Q]_\sim$, are events. The equivalence classes of backwards transitions, written $[\mathbf{H}_P \vdash P \overset{\alpha_0}{\underset{u_0,D_0}{\rightsquigarrow}} \mathbf{H}_Q \vdash Q]_\sim$ are reverse events.

Lemma 2.17 (Coinitial propagation of independence [19]). *Given a diamond consisting of transitions* $\mathbf{H}_P \vdash P \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_Q \vdash Q$ *and* $\mathbf{H}_P \vdash P \xrightarrow[u_1,D_1]{\alpha_1} \mathbf{H}_R \vdash R$ *and* $\mathbf{H}_Q \vdash Q \xrightarrow[u_1,D_1]{\alpha_1} \mathbf{H}_S \vdash S$ *and* $\mathbf{H}_R \vdash R \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_S \vdash S$ *with* $\mathbf{H}_P \vdash P \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_Q \vdash Q$ *and* $\mathbf{H}_P \vdash P \xrightarrow[u_1,D_1]{\alpha_1} \mathbf{H}_R \vdash R$ *independent, we get* $\mathbf{H}_Q \vdash Q \xrightarrow[u_1,D_1]{\alpha_1} \mathbf{H}_S \vdash S$ *and* $\mathbf{H}_Q \vdash Q \overset{\alpha_0}{\underset{u_0,D_0}{\rightsquigarrow}} \mathbf{H}_P \vdash P$ *are independent.*

PROOF. Follows from independence being defined on labels of transitions.

Lemma 2.18 (Independence respects events [19]). *Whenever* $\mathbf{H}_P \vdash P \overset{\alpha_0}{\underset{u_0,D_0}{\rightarrowtail}} \mathbf{H}_Q \vdash Q \sim \mathbf{H}_{P'} \vdash P' \overset{\alpha_0}{\underset{u_0,D_0}{\rightarrowtail}} \mathbf{H}_{Q'} \vdash Q'$ *and* $\mathbf{H}_{P'} \vdash P' \overset{\alpha_0}{\underset{u_0,D_0}{\rightarrowtail}} \mathbf{H}_{Q'} \vdash Q'$ *and* $\mathbf{H}_R \vdash R \overset{\alpha_1}{\underset{u_1,D_1}{\rightarrowtail}} \mathbf{H}_S \vdash S$ *are independent, we get* $\mathbf{H}_P \vdash P \overset{\alpha_0}{\underset{u_0,D_0}{\rightarrowtail}} \mathbf{H}_Q \vdash Q$ *and* $\mathbf{H}_R \vdash R \overset{\alpha_1}{\underset{u_1,D_1}{\rightarrowtail}} \mathbf{H}_S \vdash S$ *are independent.*

PROOF. Follows from independence being defined on labels of transitions.

These lemmas along with Proposition 2.10 and Lemmas 2.12 and 2.13 give πIH a number of key properties from [19], including causal safety (Proposition 2.20) and causal liveness (Proposition 2.21). To define these properties, we first define how to count occurrences in a path.

Definition 2.19 ([19]). Let t be a trace and e be an event. Let $\sharp\,(t, e)$ be the number of occurrences of transitions $\mathbf{H}_P \vdash P \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_Q \vdash Q$ in t such that $\mathbf{H}_P \vdash P \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_Q \vdash Q \in e$, minus the number of occurrences of transitions $\mathbf{H}_R \vdash R \overset{\alpha_1}{\underset{u_1,D_1}{\rightsquigarrow}} \mathbf{H}_S \vdash S$ such that $\mathbf{H}_S \vdash S \xrightarrow[u_1,D_1]{\alpha_1} \mathbf{H}_R \vdash R \in e$.

Proposition 2.20 (Causal safety [19]). *Whenever we have a transition* $\mathbf{H}_P \vdash P \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_Q \vdash Q$ *and a trace* $t = \mathbf{H}_Q \vdash Q \rightarrowtail^* \mathbf{H}_R \vdash R$ *such that* $\sharp\,(t, [\mathbf{H}_P \vdash P \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_Q \vdash Q]_\sim) = 0$*, and a transition* $\mathbf{H}_S \vdash S \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_R \vdash R$ *with* $\mathbf{H}_P \vdash P \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_Q \vdash Q \sim \mathbf{H}_S \vdash S \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_R \vdash R$*, we get that all* $\mathbf{H}_{P'} \vdash P' \overset{\alpha_1}{\underset{u_1,D_1}{\rightarrowtail}} \mathbf{H}_{Q'} \vdash Q'$ *such that* $\sharp\,(t, [\mathbf{H}_{P'} \vdash P' \overset{\alpha_1}{\underset{u_1,D_1}{\rightarrowtail}} \mathbf{H}_{Q'} \vdash Q']_\sim) > 0$*, are independent of* $\mathbf{H}_P \vdash P \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_Q \vdash Q$.

PROOF. Follows from Proposition 2.10 and Lemmas 2.12, 2.13, 2.17, and 2.18.

Proposition 2.21 (Causal liveness [19]). *When we have a transition* $\mathbf{H}_P \vdash P \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_Q \vdash Q$ *and a trace* $t = \mathbf{H}_Q \vdash Q \rightarrowtail^* \mathbf{H}_R \vdash R$ *such that* $\sharp\,(t, [\mathbf{H}_P \vdash P \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_Q \vdash Q]_\sim) = 0$*, and all* $\mathbf{H}_{P'} \vdash P' \overset{\alpha_1}{\underset{u_1,D_1}{\rightarrowtail}} \mathbf{H}_{Q'} \vdash Q'$ *such that* $\sharp\,(t, [\mathbf{H}_{P'} \vdash P' \overset{\alpha_1}{\underset{u_1,D_1}{\rightarrowtail}} \mathbf{H}_{Q'} \vdash Q']_\sim) > 0$*, are independent of* $\mathbf{H}_P \vdash P \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_Q \vdash Q$*, then we get a transition* $\mathbf{H}_S \vdash S \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_R \vdash R$ *with* $\mathbf{H}_P \vdash P \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_Q \vdash Q \sim \mathbf{H}_S \vdash S \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_R \vdash R$.

PROOF. Follows from Proposition 2.10 and Lemmas 2.12, 2.13, 2.17, and 2.18.

The last result required by [19] to get all the properties defined in their paper is that independence of events is coinitial. However, our definition of independence allows a process to have independent transitions without having a state in which both can be performed. Consider the process $(\emptyset, \emptyset, \emptyset) \vdash \overline{a}(b).(\overline{c}(d) \mid 0) + \overline{e}(f).(0 \mid \overline{g}(h))$. This process will eventually be able to do $\xrightarrow[0[\overline{c}(d)][0],\emptyset]{\overline{c}(d)}$ or $\xrightarrow[1[\overline{g}(h)][0],\emptyset]{\overline{g}(h)}$, which are independent transitions, but no state of P exists where both can be performed. We therefore cannot say that independence of events is coinitial in πIH. Allowing actions on opposite

sides of a choice to be independent sets us apart from most traditional definitions of independence or concurrency, such as [3]. However, since these independent events on opposite sides of a choice are caused by conflicting events due to guarded choice, and are therefore not coinitial or consecutive, this does not cause us any problems.

Independence of events being coinitial is used in [19] to prove that reversing preserves independence.

Lemma 2.22 (Reversing preserves independence).

- *Given independent transitions* $\mathbf{H}_P \vdash P \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_Q \vdash Q$ *and* $\mathbf{H}_{P'} \vdash P' \overset{\alpha_1}{\underset{u_1,D_1}{\rightarrowtail}} \mathbf{H}_{Q'} \vdash Q'$*, we also have independence between* $\mathbf{H}_Q \vdash Q \overset{\alpha_0}{\underset{u_0,D_0}{\rightsquigarrow}} \mathbf{H}_P \vdash P$ *and* $\mathbf{H}_{P'} \vdash P' \overset{\alpha_1}{\underset{u_1,D_1}{\rightarrowtail}} \mathbf{H}_{Q'} \vdash Q'$.
- *Given independent transitions* $\mathbf{H}_P \vdash P \overset{\alpha_0}{\underset{u_0,D_0}{\rightsquigarrow}} \mathbf{H}_Q \vdash Q$ *and* $\mathbf{H}_{P'} \vdash P' \overset{\alpha_1}{\underset{u_1,D_1}{\rightarrowtail}} \mathbf{H}_{Q'} \vdash Q'$*, we also have independence between* $\mathbf{H}_Q \vdash Q \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_P \vdash P$ *and* $\mathbf{H}_{P'} \vdash P' \overset{\alpha_1}{\underset{u_1,D_1}{\rightarrowtail}} \mathbf{H}_{Q'} \vdash Q'$.

PROOF. Follows from independence being defined on labels of transitions.

Since we based our calculus on the π-calculus presented in [14], we would like to show a correspondence between the two. Our calculus is less expressive because it is a πI-calculus rather than a π-calculus, but contains more information in the histories because it is able to reverse. We show that by removing the extra information needed to reverse from our extrusion histories, we can get a fragment of [14].

Definition 2.23 (Mapping from πIH to π-calculus with extrusion histories). Given πIH process $\mathbf{H} \vdash P$, the corresponding π-calculus process with extrusion history, $I(\mathbf{H} \vdash P)$, is defined as:

$$I(\mathbf{H} \vdash P) = (I(\overline{H}), I(\underline{H})) \vdash I(P) \qquad I(\{(\overline{a}(b), u)\} \cup \overline{H}) = \{(b, u)\} \cup I(\overline{H})$$
$$I(\{(a(b), u)\} \cup \underline{H}) = \{(b, u)\} \cup I(\underline{H}) \qquad I(P_0|P_1) = I(P_0)|I(P_1)$$
$$I(\sum_{i\in J} P_i) = \sum_{i\in J} I(P_i) \qquad I((\nu a)P) = (\nu a)I(P)$$
$$I(\overline{a}(b).P) = (\nu b)\overline{a}\,\langle b\rangle\,.I(P) \qquad I(a(x).P) = \underline{a}(x).I(P)$$
$$I(!P) = !I(P)$$

Proposition 2.24. *Given a πIH process* $\mathbf{H} \vdash P$*:*

- *if there exists a transition* $\mathbf{H} \vdash P \xrightarrow[u]{\mu} \mathbf{H}' \vdash P'$*, then there exists a transition* $I(\mathbf{H} \vdash P) \xrightarrow[u]{\mu} I(\mathbf{H}'' \vdash P'')$ *for some* $\mathbf{H}'' \vdash P'' \equiv \mathbf{H}' \vdash P'$ *in the semantics of [14];*

- *if* $I(\mathbf{H} \vdash P) \xrightarrow[u]{\mu} I(\mathbf{H}' \vdash P')$, *then* $\mathbf{H} \vdash P \xrightarrow[u]{\mu} \mathbf{H}' \vdash P'$.

PROOF. The only difference between the semantics is outputs being treated as bound names, and therefore put in extrusion histories automatically, adding a history of communications, which I removes, and replacing the (Rep) rule with structural congruence.

3. πI-calculus reversible semantics with annotations

In this section we introduce πIK, our statically reversible πI-calculus. Unlike πIH introduced in the previous section, πIK's approach to reversibility is based on previous statically reversible calculi [21, 25]. Both of these papers use *communication keys* to denote past actions and which other actions they have interacted with, so $a(x)|\overline{a}(b) \xrightarrow{\tau[n]} a(b)[n]|\overline{a}(b)[n]$ means a communication with the key n has taken place between the two actions. We apply this idea to define early semantics of πIK, which has the following syntax:

$$P ::= \alpha.P \mid \alpha[n].P \mid P_0 + P_1 \mid P_0|P_1 \mid (\nu x)P \mid !P \quad \alpha ::= \overline{a}(b) \mid a(b)$$

The primary difference between applying communication keys to CCS and the πI-calculus is the need to deal with substitution. We need to keep track of not only which actions have communicated with each other, but also which names were substituted when. We do this by giving the substituted names a key, $a_{[n]}$, but otherwise treating them the same as those without the key, except when undoing the input associated with n. By restricting ourselves to the πI-calculus, we avoid using the keys to keep track of opened restrictions as in [21].

Example 3.1 (Substitution). *The process* $a(x).\overline{x}(c)|\overline{a}(b)$ *can perform a communication of a with key* n*:*

$$a(x).\overline{x}(c)|\overline{a}(b) \xrightarrow{\tau[n]} (\nu b)(a(b)[n].\overline{b_{[n]}}(c)|\overline{a}(b)[n])$$

This assigns the key n *to both parts of the communication, just as in CCSK [25], and additionally assigns* n *to all the names that were substituted during this communication. After performing the substitution, the process does not remember the variable used in the input. Therefore, when reversing, the process simply picks a new one and substitute that for any occurrences of* $b_{[n]}$*:*

$$(\nu b)(a(b)[n].\overline{b_{[n]}}(c)|\overline{a}(b)[n]) \overset{\tau[n]}{\rightsquigarrow} a(y).\overline{y}(c)|\overline{a}(b)$$

Table 3 shows the forward semantics of the πIK-calculus. The reverse semantics can be seen in Table 4. We use α to range over input and output actions and μ over input, output, and τ. We use $\rightsquigarrow$ to denote reverse actions, $\mathsf{std}(P)$ to denote that P is a *standard process*, meaning it does not contain any past actions, $\mathsf{keys}(P)$ to denote the keys mentioned in P, and $\mathsf{fsh}[n](P)$ to denote that a key n is fresh for P. The semantics use structural congruence as defined in Table 5. Our semantics very much resemble

$$\frac{\mathsf{std}(P) \quad P' = P[x := b_{[n]}]}{a(x).P \xrightarrow{a(b)[n]} a(b)[n].P'} \qquad \frac{\mathsf{std}(P)}{\overline{a}(b).P \xrightarrow{\overline{a}(b)[n]} \overline{a}(b)[n].P}$$

$$\frac{P \xrightarrow{\mu[m]} P' \quad m \neq n \quad \text{if } \mu = \overline{a}(x) \text{ then } x \notin n(\alpha)}{\alpha[n].P \xrightarrow{\mu[m]} \alpha[n].P'}$$

$$\frac{P_0 \xrightarrow{\mu[n]} P_0' \quad \mathsf{fsh}[n](P_1) \quad \text{if } \mu = \overline{a}(b) \text{ then } b \notin \mathsf{fn}(P_1)}{P_0|P_1 \xrightarrow{\mu[n]} P_0'|P_1} \qquad \frac{P_0 \xrightarrow{\mu[n]} P_0' \quad \mathsf{std}(P_1)}{P_0 + P_1 \xrightarrow{\mu[n]} P_0' + P_1}$$

$$\frac{P_0 \xrightarrow{a(b)[n]} P_0' \quad P_1 \xrightarrow{\overline{a}(b)[n]} P_1' \quad b \notin \mathsf{fn}(P_0)}{P_0|P_1 \xrightarrow{\tau[n]} (\nu b)(P_0'|P_1')} \qquad \frac{\mathsf{std}(P)}{\overline{a}(b).P \xrightarrow{\overline{a}(b)[n]} \overline{a}(b)[n].P}$$

$$\frac{P \xrightarrow{\mu[m]} P' \quad a \notin n(\mu)}{(\nu a)P \xrightarrow{\mu[m]} (\nu a)P'} \qquad \frac{P \equiv Q \xrightarrow{\mu[n]} Q' \equiv P'}{P \xrightarrow{\mu[n]} P'}$$

Table 3: πIK-calculus forward semantics

$$\frac{\mathsf{std}(P) \quad x \notin n(P) \quad P' = P[b_{[m]} := x]}{a(b)[m].P \overset{a(b)[m]}{\rightsquigarrow} a(x).P'} \qquad \frac{\mathsf{std}(P)}{\overline{a}(b)[n].P \overset{\overline{a}(b)[n]}{\rightsquigarrow} \overline{a}(b).P} \qquad \frac{P \overset{\mu[m]}{\rightsquigarrow} P' \quad m \neq n}{\alpha[n].P \overset{\mu[m]}{\rightsquigarrow} \alpha[n].P'}$$

$$\frac{P_0 \overset{\mu[n]}{\rightsquigarrow} P_0' \quad \mathsf{fsh}[n](P_1) \quad \text{if } \mu = \overline{a}(b) \text{ then } b \notin \mathsf{fn}(P_1)}{P_0|P_1 \overset{\mu[n]}{\rightsquigarrow} P_0'|P_1} \qquad \frac{P_0 \overset{a(b)[n]}{\rightsquigarrow} P_0' \quad P_1 \overset{\overline{a}(b)[n]}{\rightsquigarrow} P_1'}{(\nu b)(P_0|P_1) \overset{\tau[n]}{\rightsquigarrow} P_0'|P_1'}$$

$$\frac{P_0 \overset{\mu[n]}{\rightsquigarrow} P_0' \quad \mathsf{std}(P_1)}{P_0 + P_1 \overset{\mu[n]}{\rightsquigarrow} P_0' + P_1} \qquad \frac{P \overset{\mu[m]}{\rightsquigarrow} P' \quad b \notin n(\mu)}{(\nu b)P \overset{\mu[m]}{\rightsquigarrow} (\nu b)P'} \qquad \frac{P \equiv Q \overset{\mu[n]}{\rightsquigarrow} Q' \equiv P'}{P \overset{\mu[n]}{\rightsquigarrow} P'}$$

Table 4: πIK reverse semantics

those of CCSK, with the exceptions of substitution and ensuring that any name being output does not appear elsewhere in the process.

The reason we need to consider substitution despite dealing with πI-calculus is that we are describing early semantics. In early semantics, unlike late, an input can receive a name from the environment that is also a free name of the process.

Example 3.2. *The process $a(x).\overline{b}(c).x(y)$ can, since b is free, perform a transition*

$$a(x).\overline{b}(c).x(y) \xrightarrow{a(b)[n]} a(b)[n].\overline{b}(c).b_{[n]}(y)$$

In this case, when reversing the input, we would not want to substitute the b doing the output.

We show that forwards and reverse transitions are inverse of one another in Proposition 3.5. We apply the same notions of reachability here as in the previous section, but using a standard process instead of a process with empty histories.

$P\|0 \equiv P$	$P_0\|P_1 \equiv P_1\|P_0$	$P_0\|(P_1\|P_2) \equiv (P_0\|P_1)\|P_2$
$P + 0 \equiv P$	$P_0 + P_1 \equiv P_1 + P_0$	$P_0 + (P_1 + P_2) \equiv (P_0 + P_1) + P_2$
$!P \equiv !P\|P$	$(\nu x)(\nu y)P \equiv (\nu y)(\nu x)P$	$(\nu a)(P_0\|P_1) \equiv ((\nu a)P_0\|P_1)$ if $a \notin n(P_1)$

Table 5: Structural congruence

Lemma 3.3. *Given a forwards reachable process P, any subprocess of P, P' is also forwards reachable.*

PROOF. By straightforward induction on P.

Lemma 3.4. *Given a forwards reachable process P, if $P \xrightarrow{\overline{a}(x)[n]}$ then there cannot exist a past output action $\overline{b}(x)[m]$ anywhere in P.*

PROOF. See Appendix C.1.

Proposition 3.5 (Loop).

1. *Given a process P, if $P \xrightarrow{\mu[n]} Q$ then $Q \overset{\mu[n]}{\rightsquigarrow} P$.*
2. *Given a forwards reachable process P, if $P \overset{\mu[n]}{\rightsquigarrow} Q$ then $Q \xrightarrow{\mu[n]} P$.*

PROOF. We prove this by induction on the transitions in Appendix C.2.

We also have a diamond confluence property for reverse transitions. This means we can reverse any independent actions in whichever order we want and always get the same result.

Proposition 3.6 (Reverse diamond). *Given forwards reachable processes P, Q, and R, if $P \overset{\mu[m]}{\rightsquigarrow} Q$ and $P \overset{\mu'[n]}{\rightsquigarrow} R$ and $m \neq n$, then there exists a process S such that $Q \overset{\mu'[n]}{\rightsquigarrow} S$ and $R \overset{\mu[m]}{\rightsquigarrow} S$.*

PROOF. We use structural induction on P to prove both these at once in Appendix C.3.

We also show that whenever we have a reverse transition for a key, every reverse transition with that key has the same label and structurally congruent end processes. This means we cannot e.g. reverse one part of a communication without reversing the other, or get two different results from reversing the same action.

Proposition 3.7. *Given forwards reachable processes P, Q, and R, if $P \overset{\mu[m]}{\rightsquigarrow} Q$ and $P \overset{\mu'[m]}{\rightsquigarrow} R$ then $\mu = \mu'$ and $R \equiv Q$.*

PROOF. We prove this by structural induction in Appendix C.4.

We also have what Danos and Krivine [10] refer to as a parabolic trace, meaning a trace wherein all reverse transitions happen before any forward transitions, whenever we have a trace between two processes.

Lemma 3.8. *Given a forwards-reachable process P, if $P \overset{\mu[m]}{\rightsquigarrow} Q$ then Q is forwards-reachable.*

PROOF. Since P is forwards-reachable, by Proposition 3.5, we have a process P' such that $P \rightsquigarrow^* P'$ and $\mathsf{std}(P')$. By repeated use of Proposition 3.6, we get $Q \rightsquigarrow^* P'$. Therefore, by Proposition 3.5, Q is forwards-reachable.

Theorem 3.9 (Parabola). *Given processes P and Q, such that $P \rightarrowtail^* Q$, there exists a process R such that $P \rightsquigarrow^* R \to^* Q$.*

PROOF. See Appendix C.5.

Corollary 3.10. *A process P is reachable if and only if it is forwards-reachable.*

This means all reachable processes are forwards-reachable, and we can therefore take any previous results about forwards-reachable processes and apply them to all reachable processes. We also get a forwards diamond confluence (Proposition 3.13), similar to the reverse diamond in Proposition 3.6. We can view actions as being independent if the resulting processes are not structurally congruent but can still eventually reach the same state.

Lemma 3.11. *Given a process P, if $P \xrightarrow{\mu[m]}$ then $m \notin \mathsf{keys}(P)$*

PROOF. Straightforward from the semantics.

Lemma 3.12. *Given processes P, Q, if $P \to^* Q$ then $\mathsf{keys}(P) \subseteq \mathsf{keys}(Q)$.*

PROOF. Straightforward from the semantics.

Proposition 3.13 (Forward diamond [25]).

1. *Given reachable processes P, Q, and R, if $P \xrightarrow{\mu[m]} Q$, $P \xrightarrow{\mu'[n]} R$, $m = n$, and there exists a process T such that $Q \to^* T$ and $R \to^* T$, then $\mu = \mu'$ and $R \equiv Q$.*
2. *Given reachable processes P, Q, and R, if $P \xrightarrow{\mu[m]} Q$, $P \xrightarrow{\mu'[n]} R$, and there exists a process T such that $Q \to^* T$ and $R \to^* T$, then there exists a process S such that $Q \xrightarrow{\mu'[n]} S$ and $R \xrightarrow{\mu'[m]} S$ and $S \to^* T$.*

PROOF. We prove both simultaneously by structural induction on P in Appendix C.6.

4. Mapping from πIH to πIK

We will now define a mapping from πIH to πIK and show that we have an operational correspondence in Theorem 4.8. The extrusion histories store more information than the keys, as they keep track of which names were substituted, as illustrated by Example 4.1. This means we lose some information in our mapping, but not information we need.

Example 4.1. *Consider the processes* $(\emptyset, \{(a(b), [a(x)][0])\}, \emptyset) \vdash 0$ *and* $a(b)[n]$*. These are the result of* $a(x)$ *receiving* b *in the two different semantics. We can see that the extrusion history remembers that the input name was* x *before* b *was received, but the keys do not remember, and when reversing the action could use any name as the input name. This does not make a great deal of difference, as after reversing* $a(b)$*, the process with the extrusion history can also* α*-convert* x *to any name.*

Since we intend to define a mapping from processes with extrusion histories to processes with keys, we first describe how to add keys to substituted names in a process in Definition 4.2. The function S takes a process, P_1, in which we will be adding keys to the previously substituted names, the key we want to add, $[n]$, the name x which has been substituted and we want to add keys to, and P_2, a previous state of P_1 where the substitution has not yet taken place. We use P_2 to distinguish instances of x created by the substitution we are currently marking from instances of x which existed before the substitution took place.

Definition 4.2 (Substituting in πIK-process). Given a πIK process P_1, a πI-calculus process without keys, P_2, a key n, and a name x, we can add the key n to any names which x has been substituted with, by applying $S(P_1, P_2, [n], x)$, defined as:

1. $S\,(0, 0, [n], x) = 0$
2. $S\left(\sum_{i\in I} P_{i1}, \sum_{i\in I} P_{i2}, [n], x\right) = \sum_{i\in I} S\left(P_{i1}, P_{i2}, [n], x\right)$
3. $S\left(P_1|Q_1, P_2|Q_2, [n], x\right) = S\left(P_1, P_2, [n], x\right)|S\left(Q_1, Q_2, [n], x\right)$
4. $S\left((\nu a)P_1, (\nu b)P_2, [n], x\right) = P_1'$ where:
   if $x = b$ then $P_1' = (\nu a)P_1$ and otherwise $P_1' = (\nu a)S\left(P_1, P_2, [n], x\right)$.
5. $S\left(\alpha_1.P_1, \alpha_2.P_2, [n], x\right) = \alpha_1'.P_1'$ where:
   if $\alpha_2 \in \{x(c), \overline{x}(c)\}$ then $\alpha_1' = \alpha_{1_{[n]}}$ and otherwise $\alpha_1' = \alpha_1$;
   if $\alpha_2 \in \{c(x), \overline{c}(x)\}$ then $P_1' = P_1$ and otherwise $P_1' = S\left(P_1, P_2, [n], x\right)$.
6. $S\left(\alpha_1[m].P_1, \alpha_2.P_2, [n], x\right) = \alpha_1'[m].P_1'$ where:
   if $\alpha_2 \in \{x(c), \overline{x}(c)\}$ then $\alpha_1' = \alpha_{1_{[n]}}$ and otherwise $\alpha_1' = \alpha_1$;
   if $\alpha_2 \in \{c(x), \overline{c}(x)\}$ then $P_1' = P_1$ and otherwise $P_1' = S\left(P_1, P_2, [n], x\right)$.
7. $S\left(!P_1, !P_2, [n], x\right) = !S\left(P_1, P_2, [n], x\right)$
8. $S\left(P_1|P_1', !P_2, [n], x\right) = S\left(P_1, !P_2, [n], x\right)|S\left(P_1', P_2, [n], x\right)$

9. $S\left(!P_1, P_2|P_2', [n], x\right) = S\left(!P_1, P_2, [n], x\right)|S\left(P_1, P_2', [n], x\right)$

where $a(b)_{[n]} = a_{[n]}(b)$ and $\overline{a}(b)_{[n]} = \overline{a_{[n]}}(b)$

We also define the root of a πIK process as removing all keys from the process.

Definition 4.3 (Root). We say that a πIK process, P, has a root, $\mathsf{rt}(P)$, defined as:

$$\mathsf{rt}(0) = 0 \quad \mathsf{rt}(!P) = \mathsf{rt}(P) \quad \mathsf{rt}(P_0|P_1) = \mathsf{rt}(P_0)|\mathsf{rt}(P_1) \quad \mathsf{rt}(\alpha.P) = \alpha.\mathsf{rt}(P)$$
$$\mathsf{rt}(P_0 + P_1) = \mathsf{rt}(P_0) + \mathsf{rt}(P_1) \quad \mathsf{rt}(\alpha[m].P) = \alpha.\mathsf{rt}(P) \quad \mathsf{rt}((\nu x)P) = (\nu x)\mathsf{rt}(P)$$

In Lemma 4.4 we demonstrate that S does indeed annotate any name, which was was substituted for x_1, with n.

Lemma 4.4. *Given a standard πIK-calculus process P, a πIK-calculus process P', a series of substitutions $[x_1 := a_1][x_2 := a_2] \dots [x_k := a_k]$, such that $\mathsf{rt}(P') \equiv P[x_1 := a_1][x_2 := a_2] \dots [x_k := a_k]$ using the definition of $\equiv$ from Section 2, and a key $[n]$, we get $S(P', P, [n], x_1) = P''$ for some P'' such that $\mathsf{rt}(P'') \equiv P[x_1 := a_{1_{[n]}}][x_2 := a_2] \dots [x_k := a_k]$.*

PROOF. We prove this by structural induction on P in Appendix D.1.

Being able to annotate our names with keys, we can define a mapping, E, from extrusion histories to keys in Definition 4.6, thereby creating a πIK process which has performed the same actions as the πIH process. This is done by E iterating over the extrusions, having one process which builds πIK-process, and another that keeps track of which state of the original πIH process has been reached. When turning an extrusion into a keyed action, we use the locations as key and also give each extrusion an extra copy of its location to use for determining where the action came from. This way we can use one copy to iteratively go through the process, removing splits from the path as we go through them, while still having another intact copy of the location to use as the final key. If E encounters a parallel composition (case 2), it splits its extrusion histories in three. One part, $\mathbf{H}_{\mathsf{shared}}$ contains the locations which have an empty path, and therefore belong to actions from before the processes split. Another part contains the locations beginning with 0, and goes to the first part of the process. And finally the third part contains the locations beginning with 1, and goes to the second part of the process. When turning an input memory from the history into a past input action in the process (case 4), we use S (Definition 4.2) to add keys to the substituted names. When E encounters a restriction (case 5), it iteratively moves a memory that can be used inside the restriction inside. It does this iteratively until there are no such memories left in the extrusion histories. We apply E to a process in Example 4.7.

Definition 4.5. The function lcopy gives each member of an extrusion history an extra copy of its location:

$\mathsf{lcopy}(H^*) = \{(\mu, u, u) \mid (\mu, u) \in H^*\}$

$\mathsf{lcopy}(\overline{H}, \underline{H}, H) = (\mathsf{lcopy}(\overline{H}), \mathsf{lcopy}(\underline{H}), \mathsf{lcopy}(H))$

Definition 4.6. Given a πIH process, $\mathbf{H} \vdash P$, we can create an equivalent πIK process, $E(\mathsf{lcopy}(\mathbf{H}) \vdash P, P) = P'$ defined as

1. $E((\emptyset,\emptyset,\emptyset) \vdash P, P') = P$
2. $E(\mathbf{H} \vdash P_0|P_1, P_0'|P_1') = E(\mathbf{H}_{\mathsf{shared}} \vdash P_0''|P_1'', P_0'''|P_1''')$ where:
   $\mathbf{H}_{\mathsf{shared}} = (\{(\alpha,u,u') \mid (\alpha,u,u') \in \overline{H} \text{ and } u \neq iu''\}, \{(\alpha,u,u') \mid (\alpha,u,u') \in \underline{H} \text{ and } u \neq iu''\}, \emptyset)$
   $P_0'' = E((\overline{H_0}, \underline{H_0}, H_0) \vdash P_0, P_0')$ where:
   $\overline{H_0} = \{(\overline{a}(b), u_0, u_0') \mid (\overline{a}(b), 0u_0, u_0') \in \overline{H} \text{ or } ((\overline{a}(b), \alpha_1), \langle 0u_0, 1u_1\rangle, u_0') \in H\}$
   $\underline{H_0} = \{(a(b), u_0, u_0') \mid (a(b), 0u_0, u_0') \in \underline{H} \text{ or } ((a(b), \alpha_1), \langle 0u_0, 1u_1\rangle, u_0') \in H\}$
   $H_0 = \{((\alpha,\alpha'), u, u') \mid ((\alpha,\alpha'), 0u, u') \in H\}$
   $P_1'' = E((\overline{H_1}, \underline{H_1}, H_1) \vdash P_1, P_1'))$ where:
   $\overline{H_1} = \{(\overline{a}(b), u_1, u_1') \mid (\overline{a}(b), 1u_1, u_1') \in \overline{H} \text{ or } ((\alpha_0, \overline{a}(b)), \langle 0u_0, 1u_1\rangle, u_1') \in H\}$
   $\underline{H_1} = \{(a(b), u_1, u_1') \mid (a(b), 1u_1, u_1') \in \underline{H} \text{ or } ((\alpha_0, a(b)), \langle 0u_0, 1u_1\rangle, u_1') \in H\}$
   $H_1 = \{((\alpha,\alpha'), u, u') \mid ((\alpha,\alpha'), 1u, u') \in H\}$
   $\mathbf{H_i} \vdash P_i' \overset{\alpha_{i,0}}{\underset{u_{i,0}}{\rightsquigarrow}} \dots \overset{\alpha_{i,n}}{\underset{u_{i,n}}{\rightsquigarrow}} (\emptyset,\emptyset,\emptyset) \vdash P_i'''$ for $i \in \{0,1\}$
3. $E((\overline{H} \cup \{(\overline{a}(b), [Q][P'], u)\}, \underline{H}, H) \vdash P, P') = E(\mathbf{H} \vdash P'', Q)$
   where $P'' = \overline{a}(b)\,[u]\,.P + \sum_{i \in I \setminus \{j\}} \alpha_i.P_i$
   if $Q = \sum_{i\in I} \alpha_i.P_i$, $\overline{a}(b) = \alpha_j$, and $P' = P_j$
4. $E((\overline{H}, \underline{H} \cup \{(a(b), [Q][P'], u)\}, H) \vdash P, P') = E(\mathbf{H} \vdash a(b)\,[u]\,.S(P, P_j, [u], x) + \sum_{i \in I \setminus \{j\}} \alpha_i.P_i, Q)$
   if $Q = \sum_{i\in I} \alpha_i.P_i$, $a(x) = \alpha_j$, and $P' = P_j[x := b]$
5. $E(\mathbf{H} \vdash (\nu x)P, (\nu x)P') = E(\mathbf{H} - (\alpha,u,u') \vdash P'', (\nu x)Q')$
   where $P'' = (\nu x)E((\emptyset,\emptyset,\emptyset) + (\alpha,u,u') \vdash P, P')$
   if $(\alpha,u,u') \in \overline{H} \cup \underline{H}$ and $(\emptyset,\emptyset,\emptyset) + (\alpha,u,u) \vdash P \overset{\alpha}{\underset{u}{\rightsquigarrow}} (\emptyset,\emptyset,\emptyset) \vdash Q'$
6. $E(\mathbf{H} \vdash !P, !P') = E(\mathbf{H} \vdash !P|P, !P'|P')$ if there exists $(\alpha,u,u') \in \overline{H} \cup \underline{H} \cup H$ such that $u \neq [Q][Q']$.

Example 4.7. *We will now apply E to the process*

$$(\{(\overline{b}(c), u_2)\}, \emptyset, \{((b(a), \overline{b}(a)), \langle 0u_0, 1u_1\rangle)\}) \vdash a(x) \mid 0$$

with locations $u_0 = [b(y).y(x)][a(x)]$, $u_1 = [\overline{b}(a)][0]$, and $u_2 = [\overline{b}(c).(b(y).y(x) \mid \overline{b}(a)][b(y).y(x) \mid \overline{b}(a)]$. We perform

$$E(\mathsf{lcopy}((\{(\overline{b}(c), u_2)\}, \emptyset, \{((b(a), \overline{b}(a)), \langle 0u_0, 1u_1\rangle)\})) \vdash a(x) \mid 0, a(x) \mid 0)$$

Since we are at a parallel, we use Case 2 of Definition 4.6 to split the extrusion histories and processes into three to get $E((\{(\overline{b}(c), u_2, u_2)\}, \emptyset, \emptyset) \vdash P_0 \mid P_1, b(y).y(x) \mid \overline{b}(a))$

and look separately at the parallel components $P_0 = E((\emptyset, \{(b(a), u_0, \langle 0u_0, 1u_1\rangle)\}, \emptyset) \vdash a(x), a(x))$ *and* $P_1 = E((\{(\overline{b}(a), u_1, \langle 0u_0, 1u_1\rangle)\}, \emptyset, \emptyset) \vdash 0, 0)$.

To find P_0*, we look at* u_0*, and find that it has* $a(x)$ *as its result, meaning we can apply Case 4. This gives us*

$$E((\emptyset, \emptyset, \emptyset) \vdash b(a)[\langle 0u_0, 1u_1\rangle].S(a(x), y(x), [\langle 0u_0, 1u_1\rangle], y), b(y).y(x))$$

And by applying Case 5 of Definition 4.2, $S(a(x), y(x), [\langle 0u_0, 1u_1\rangle], y) = a_{[\langle 0u_0, 1u_1\rangle]}(x)$. *Since we have no more extrusions to add, we apply Case 1 to get our process* $P_0 = b(a)[\langle 0u_0, 1u_1\rangle].a_{[\langle 0u_0, 1u_1\rangle]}(x)$.

To find P_1*, we similarly look at* u_1 *and find that we can apply Case 3. This gives us* $P_1 = \overline{b}(a)[\langle 0u_0, 1u_1\rangle].0$.

We can then apply Case 3 to $E((\{(\overline{b}(c), u_2, u_2)\}, \emptyset, \emptyset) \vdash P_0 \mid P_1, b(y).y(x) \mid \overline{b}(a))$. *This gives us our final process,*

$$\overline{b}(c)[u_2].b(a)[\langle 0u_0, 1u_1\rangle].a_{[\langle 0u_0, 1u_1\rangle]}(x) \mid \overline{b}(a)[\langle 0u_0, 1u_1\rangle].0$$

We can then show, in Theorem 4.8, that we have an operational correspondence between our two calculi and E preserves transitions. Item 1 states that every transition in πIH corresponds to one in πIk process generated by E, and Item 2 vice versa.

Theorem 4.8. *Given a reachable* πIH *process,* $\mathbf{H} \vdash P$*, and an action,* μ*,*

1. *if there exists a location* u *such that* $\mathbf{H} \vdash P \overset{\mu}{\underset{u}{\rightarrowtail}} \mathbf{H}' \vdash P'$ *then we get a transition* $E(\mathsf{lcopy}(\mathbf{H}) \vdash P, P) \overset{\mu[u]}{\longrightarrowtail} E(\mathsf{lcopy}(\mathbf{H}') \vdash P', P')$*;*
2. *if* $E(\mathsf{lcopy}(\mathbf{H}) \vdash P, P) \overset{\mu[n]}{\longrightarrowtail} Q$ *then there exists a location,* u*, and a* πIH *process,* $\mathbf{H}' \vdash P'$*, such that* $\mathbf{H} \vdash P \overset{\mu}{\underset{u}{\rightarrowtail}} \mathbf{H}' \vdash P'$ *and* $P'' \equiv E(\mathsf{lcopy}(\mathbf{H}') \vdash P', P')$.

PROOF. We prove both these properties by induction on the size of $\overline{H} \cup \underline{H} \cup H$ and the structure of P in Appendix D.2.

A correspondence between CCSK and RCCS was used to transfer results between the two calculi in [16]. Our correspondence is not as strong as theirs, since we cannot get the location (or set of causes) of a πIH transition from a πIK transition. We can however use our correspondence to transfer results from πIK to πIH. For example, the forward diamond property of Proposition 3.13, implies Proposition 4.9.

Proposition 4.9 (πIH forward diamond). *Given reachable processes* $\mathbf{H}_P \vdash P$, $\mathbf{H}_Q \vdash Q$*, and* $\mathbf{H}_R \vdash R$*, if* $\mathbf{H}_P \vdash P \xrightarrow[u_0]{\mu_0} \mathbf{H}_Q \vdash Q$, $\mathbf{H}_P \vdash P \xrightarrow[u_1]{\mu_1} \mathbf{H}_R \vdash R$*, and there exists a process* $\mathbf{H}_T \vdash T$ *such that* $\mathbf{H}_Q \vdash Q \rightarrow^* \mathbf{H}_T \vdash T$ *and* $\mathbf{H}_R \vdash R \rightarrow^* \mathbf{H}_T \vdash T$*, then there exists processes* $\mathbf{H}_S \vdash S$ *and* $\mathbf{H}'_S \vdash S'$ *such that* $\mathbf{H}_Q \vdash Q \xrightarrow[u_1]{\mu_1} \mathbf{H}_S \vdash S$ *and* $\mathbf{H}_R \vdash R \xrightarrow[u_0]{\mu_0} \mathbf{H}'_S \vdash S'$ *and* $E(\mathsf{lcopy}(\mathbf{H}_S) \vdash S, S) \equiv E(\mathsf{lcopy}(\mathbf{H}'_S) \vdash S', S')$ *and* $\mathbf{H}_S \vdash S \rightarrow^* \mathbf{H}_T \vdash T$ *and* $\mathbf{H}'_S \vdash S' \rightarrow^* \mathbf{H}_T \vdash T$.

PROOF. We prove this in Appendix C.7 using Theorem 4.8, Proposition 3.13, and Lemma 2.12.

5. Bundle event structures

In this section we will recall the definition of *labelled reversible bundle event structures* (LRBESs), which we intend to use later to define the event structure semantics of πIK and through that πIH. We also describe some operations on LRBESs, which our semantics will make use of. This section is a review of definitions from [13] (with the exception of Definition 5.3); we include the material to fix notation and make this work more self-contained. We use bundle event structures, rather than the more common prime event structures, because LRBESs yield more compact event structures with fewer events and simplifies parallel composition.

An LRBES consists of a set of events, E, a subset of which, F, are reversible, and three relations on them. The bundle relation, $\mapsto$, says that if $X \mapsto e$ then one of the events of X must have happened before e can and all events in X are in conflict with each other. The conflict relation, $\sharp$, says that if $e \sharp e'$ then e and e' cannot occur in the same configuration. The prevention relation, $\rhd$, says that if $e \rhd \underline{e'}$ then e' cannot reverse after e has happened. Since the event structure is labelled, we also have a set of labels Act, and a labelling function λ from events to labels. We use $\underline{e}$ to denote e being reversed, and e^* to denote either e or $\underline{e}$.

Definition 5.1 (Labelled Reversible Bundle Event Structure [13]). A labelled reversible bundle event structure (LRBES) is a 7-tuple $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$ where:

1. E is the set of events;
2. $F \subseteq E$ is the set of reversible events;
3. the bundle set, $\mapsto \subseteq 2^E \times (E \cup \underline{F})$, satisfies $X \mapsto e^* \Rightarrow \forall e_1, e_2 \in X . e_1 \neq e_2 \Rightarrow e_1 \sharp e_2$ and for all $e \in F$, $\{e\} \mapsto \underline{e}$;
4. the conflict relation, $\sharp \subseteq E \times E$, is symmetric and irreflexive;
5. $\rhd \subseteq E \times \underline{F}$ is the prevention relation.
6. $\lambda : E \to \mathsf{Act}$ is a labelling function.

Definition 5.2 (causal LRBES [13]). $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$ is a causal LRBES (CLRBES) if (1) if $e \rhd \underline{e'}$ then either $e \sharp e'$ or there exists an $X \subseteq E$ such that $X \mapsto e$ and $e' \in X$, (2) if $X \mapsto e$ and $e' \in X \cap F$, then $e \rhd \underline{e'}$, and (3) if $X \mapsto \underline{e}$ then $e \in X$.

An event in a LRBES can have multiple possible causes as defined in Definition 5.3. A possible cause X of an event e is a conflict-free set of events which contains a member of each bundle associated with e and contains possible causes of all events in X. This definition is not necessary for the denotational semantics, but will be useful in Section 8 for comparing the denotational and operational event structure semantics.

Definition 5.3 (Possible Cause). Given an LRBES, $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$ and an event $e \in E$, $X \subseteq E$ is a possible cause of e if

- $e \notin X$, X is finite, whenever $X' \mapsto e$ we have $X' \cap X \neq \emptyset$;
- for any $e', e'' \in \{e\} \cup X$, we have $e' \not\sharp e''$ ($X \cup \{e\}$ is conflict-free);
- for all $e' \in X$, there exists $X'' \subseteq X$, such that X'' is a possible cause of e';

- there does not exist any $X''' \subset X$, such that X''' is a possible cause of e.

Since we want to compare the event structures generated by a process to the operational semantics, we need a notion of transitions on event structures. For this purpose we use configuration systems (CSs), which event structures can be translated into.

Definition 5.4 (Configuration system [27]). A *configuration system* (CS) is a quadruple $\mathcal{C} = (E, F, \mathsf{C}, \rightarrow)$ where E is a set of events, $F \subseteq E$ is a set of reversible events, $\mathsf{C} \subseteq 2^E$ is the set of configurations, and $\rightarrow \subseteq \mathsf{C} \times 2^{E \cup \underline{F}} \times \mathsf{C}$ is a labelled transition relation such that if $X \xrightarrow{A \cup \underline{B}} Y$ then:

- $X, Y \in \mathsf{C}$, $A \cap X = \emptyset$; $B \subseteq X \cap F$; and $Y = (X \setminus B) \cup A$;
- for all $A' \subseteq A$ and $B' \subseteq B$, we have $X \xrightarrow{A' \cup \underline{B'}} Z \xrightarrow{(A \setminus A') \cup \underline{(B \setminus B')}} Y$, meaning $Z = (X \setminus B') \cup A' \in \mathsf{C}$.

Definition 5.5 (From LRBES to CS [13]). We define a mapping C_{br} from LRBESs to CSs as: $C_{br}((E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})) = (E, F, \mathsf{C}, \rightarrow)$ where:

1. $X \in \mathsf{C}$ if X is conflict-free;
2. For $X, Y \in \mathsf{C}$, $A \subseteq E$, and $B \subseteq F$, there exists a transition $X \xrightarrow{A \cup \underline{B}} Y$ if:
   (a) $Y = (X \setminus B) \cup A$; $X \cap A = \emptyset$; $B \subseteq X$; and $X \cup A$ conflict-free;
   (b) for all $e \in B$, if $e' \rhd \underline{e}$ then $e' \notin X \cup A$;
   (c) for all $e \in A$ and $X' \subseteq E$, if $X' \mapsto e$ then $X' \cap (X \setminus B) \neq \emptyset$;
   (d) for all $e \in B$ and $X' \subseteq E$, if $X' \mapsto \underline{e}$ then $X' \cap (X \setminus (B \setminus \{e\})) \neq \emptyset$.

For our semantics we need to define a prefix, restriction, parallel composition, and choice. Causal prefixing takes a label, μ, an event, e, and an LRBES, $\mathcal{E}$, and adds e to $\mathcal{E}$ with the label μ and associating every other event in $\mathcal{E}$ with a bundle containing only e. Restriction removes a set of events from an LRBES.

Definition 5.6 (Causal Prefixes [13]). Given an LRBES $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$, a label μ, and an event e, $(\mu)(e).\mathcal{E} = (E', F', \mapsto', \sharp', \rhd', \lambda', \mathsf{Act}')$ where:

1. $E' = E \cup e$;
2. $F' = F \cup e$;
3. $\mapsto' = \mapsto \cup (\{\{e\}\} \times (E \cup \{\underline{e}\}))$;
4. $\sharp' = \sharp$;
5. $\rhd' = \rhd \cup (E \times \{\underline{e}\})$;
6. $\lambda' = \lambda[e \mapsto \mu]$;
7. $\mathsf{Act}' = \mathsf{Act} \cup \{\mu\}$.

Removing a set of labels L from an LRBES removes not just events with labels in A but also events dependent on events with labels in L.

Definition 5.7 (Removing labels and their dependants). Given an LRBES $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$ and a set of labels $L \subseteq \mathsf{Act}$, we define $\rho_{\mathcal{E}}(L) = X$ as the maximum subset of E such that

1. if $e \in X$ then $\lambda(e) \notin L$;
2. if $e \in X$ then there exists a possible cause of e, x, such that $x \subseteq X$.

A choice between LRBESs puts all the events of one event structure in conflict with the events of the others.

Definition 5.8 (Choice [13]). Given LRBESs $\mathcal{E}_0, \mathcal{E}_1, \ldots, \mathcal{E}_n$, the choice between them is $\sum_{0 \leq i \leq n} \mathcal{E}_i = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$ where:

1. $E = \bigcup_{0 \le i \le n} \{i\} \times E_i$;
2. $F = \bigcup_{0 \le i \le n} \{i\} \times F_i$;
3. $X \mapsto e^*$ if $e = (i, e_i)$, $X_i \mapsto_i e_i^*$, and $X = \{i\} \times X_i$;
4. $(i, e) \sharp (j, e')$ if $i \neq j$ or $e \sharp_i e'$;
5. $(i, e) \rhd (j, e')$ if $i \neq j$ or $e \sharp_i e'$;
6. $\lambda(j, e) = \lambda_j(e)$;
7. $\mathsf{Act} = \bigcup_{0 \le i \le n} \mathsf{Act}_i$.

Definition 5.9 (Restriction [13]). Given an LRBES, $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$, restricting $\mathcal{E}$ to a set of events $E' \subseteq E$ creates $\mathcal{E} \restriction E' = (E', F', \mapsto', \sharp', \rhd', \lambda', \mathsf{Act}')$ where:

1. $F' = F \cap E'$;
2. $\mapsto' = \mapsto \cap (\mathcal{P}(E') \times (E' \cup \underline{F'}))$;
3. $\sharp' = \sharp \cap (E' \times E')$;
4. $\rhd' = \rhd \cap (E' \times \underline{F'})$;
5. $\lambda' = \lambda \restriction_{E'}$;
6. $\mathsf{Act} = \mathsf{ran}(\lambda')$.

We say that $(E', F', \mapsto', \sharp', \rhd', \lambda', \mathsf{Act}') \leq (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$.

For parallel composition we construct a product of event structures, which consists of events corresponding to synchronisations between the two event structures. The possible causes of an event (e_0, e_1) contain a possible cause of e_0 and a possible cause of e_1.

Definition 5.10 (Parallel [13]). Given two LRBESs $\mathcal{E}_0 = (E_0, F_0, \mapsto_0, \sharp_0, \rhd_0, \lambda_0, \mathsf{Act}_0)$ and $\mathcal{E}_1 = (E_1, F_1, \mapsto_1, \sharp_1, \rhd_1, \lambda_1, \mathsf{Act}_1)$, their parallel composition $\mathcal{E}_0 \times \mathcal{E}_1 = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$ with projections π_0 and π_1 where:

1. $E = E_0 \times_* E_1 = \{(e, *) \mid e \in E_0\} \cup \{(*, e) \mid e \in E_1\} \cup \{(e, e') \mid e \in E_0 \text{ and } e' \in E_1\}$;
2. $F = F_0 \times_* F_1 = \{(e, *) \mid e \in F_0\} \cup \{(*, e) \mid e \in F_1\} \cup \{(e, e') \mid e \in F_0 \text{ and } e' \in F_1\}$;
3. for $i \in \{0, 1\}$ we have $(e_0, e_1) \in E$, $\pi_i((e_0, e_1)) = e_i$;
4. for any $e^* \in E \cup \underline{F}$, $X \subseteq E$, $X \mapsto e^*$ iff there exists $i \in \{0, 1\}$ and $X_i \subseteq E_i$ such that $X_i \mapsto \pi_i(e)^*$ and $X = \{e' \in E \mid \pi_i(e') \in X_i\}$;
5. for any $e, e' \in E$, $e \sharp e'$ iff there exists $i \in \{0, 1\}$ such that $\pi_i(e) \sharp_i \pi_i(e')$, or $\pi_i(e) = \pi_i(e') \neq \bot$ and $\pi_{1-i}(e) \neq \pi_{1-i}(e')$;
6. for any $e \in E$, $e' \in F$, $e \rhd \underline{e'}$ iff there exists $i \in \{0, 1\}$ such that $\pi_i(e) \rhd_i \underline{\pi_i(e')}$;
7. $\lambda(e) = \begin{cases} \lambda_0(e_0) & \text{if } e = (e_0, *) \\ \lambda_1(e_1) & \text{if } e = (*, e_1) \\ \tau & \text{if } e = (e_0, e_1) \text{ and either } \lambda_0(e_0) = a(x) \text{ and } \lambda_1(e_1) = \overline{a}(x) \\ & \text{or } \lambda_0(e_0) = \overline{a}(x) \text{ and } \lambda_1(e_1) = a(x) \\ 0 & \text{otherwise} \end{cases}$
8. $\mathsf{Act} = \{\tau\} \cup \mathsf{Act}_0 \cup \mathsf{Act}_1$.

6. Denotational event structure semantics of πIK

In this section we define event structure semantics of πIK-calculus using a similar denotational approach to [13].

As we want to ensure that all free and bound names in our process are distinct, we modify our syntax for replication, assigning each replication an infinite set, $\mathbf{x}$, of names to substitute into the place of bound names in each created copy of the process, so that

$$!_{\mathbf{x}}P \equiv !_{\mathbf{x}\setminus\{x_0,\ldots,x_k\}}P|P\{^{x_0,\ldots,x_k}/_{a_0,\ldots,a_k}\} \text{ if } \{x_0,\ldots,x_k\} \subseteq \mathbf{x} \text{ and } \mathsf{bn}(P) = \{a_0,\ldots,a_k\}$$

Before proceeding to the semantics we also define the standard bound names of a process P, $\mathsf{sbn}(P)$, meaning the names that would be bound in P if every action was reversed, in Definition 6.1.

Definition 6.1. The standard bound names of a process P, $\mathsf{sbn}(P)$, are defined as:

$$\begin{array}{ll}
\mathsf{sbn}(a(x).P') = \{x\} \cup \mathsf{sbn}(P') & \mathsf{sbn}(a(x)[m].P') = \{x\} \cup \mathsf{sbn}(P') \\
\mathsf{sbn}(\overline{a}(x).P') = \{x\} \cup \mathsf{sbn}(P') & \mathsf{sbn}(\overline{a}(x)[m].P') = \{x\} \cup \mathsf{sbn}(P') \\
\mathsf{sbn}(P_0|P_1) = \mathsf{sbn}(P_0) \cup \mathsf{sbn}(P_1) & \mathsf{sbn}(P_0 + P_1) = \mathsf{sbn}(P_0) \cup \mathsf{sbn}(P_1) \\
\mathsf{sbn}(\nu x)P' = \{x\} \cup \mathsf{sbn}(P') & \mathsf{sbn}(!_{\mathbf{x}}P) = \mathbf{x}
\end{array}$$

We can now define the event structure semantics in Table 6. We do this using rules of the form $\{\!|P|\!\}_{(\mathcal{N},l)} = \langle \mathcal{E}, \mathsf{Init}, k\rangle$ where l is the level of unfolding of replication, $\mathcal{E}$ is an LRBES, Init is the initial configuration, $\mathcal{N} \supseteq n(P)$ is a set of names, which any input in the process could receive, and $k : \mathsf{Init} \to \mathcal{K}$ is a function assigning communication keys to the past actions, which we use in parallel composition to determine which synchronisations of past actions to put in Init. We define $\{\!|P|\!\}_{\mathcal{N}} = \sup_{l\in\mathbb{N}} \{\!|P|\!\}_{(\mathcal{N},l)}$

Most of the cases in Table 6 are straightforward uses of the RBES operators defined in Section 5. The input creates a case for each name in $\mathcal{N}$ and a choice between the cases. We have two cases for restriction, one for restriction originating from a past communication and another for restriction originating from the original process. The parallel composition is more complex as it needs to consider link causation caused by the early semantics. Each event labelled with an input of a name in standard bound names gets a bundle consisting of the event labelled with the output on that name. And each output event is prevented from reversing by the input names receiving that name.

Note that the only difference between a future and a past action is that the event corresponding to a past action is put in the initial state and given a communication key.

$$\begin{array}{ll}
\{\!|0|\!\}_{(\mathcal{N},l)} = & \langle(\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset),\emptyset,\emptyset\rangle \\
\{\!|P_0 + P_1|\!\}_{(\mathcal{N},l)} = & \langle \mathcal{E}_0 + \mathcal{E}_1, (\{0\}\times\mathsf{Init}_0) \cup (\{1\}\times\mathsf{Init}_1), k((i,e)) = k_i(e)\rangle \text{ where} \\
& \{\!|P_i|\!\} = \langle \mathcal{E}_i, \mathsf{Init}_i, k_i\rangle \text{ for } i \in \{0,1\} \\
\left\{\!\left|\overline{a}(n).P\right|\!\right\}_{(\mathcal{N},l)} = & \left\langle \overline{a}(n)(e).\mathcal{E}_P, \mathsf{Init}_P, k_P\right\rangle \text{ for some fresh } e \notin E \text{ where} \\
& \{\!|P|\!\}_{(\mathcal{N},l)} = \langle \mathcal{E}_P, \mathsf{Init}_P, k_P\rangle \\
\{\!|a(x).P|\!\}_{(\mathcal{N},l)} = & \left\langle \sum_{n\in(\mathcal{N}\setminus\mathsf{sbn}(P))} a(n)(e_n).\mathcal{E}_{P_n}, \bigcup_{n\in(\mathcal{N}\setminus\mathsf{sbn}(P))} \{n\}\times\mathsf{Init}_{P_n}, (n,e)\mapsto k_{P_n}(e)\right\rangle \\
& \text{for some fresh } e_n \notin E_n \text{ where } \{\!|P[x := n]|\!\}_{(\mathcal{N},l)} = \left\langle \mathcal{E}_{P_n}, \mathsf{Init}_{P_n}, k_{P_n}\right\rangle
\end{array}$$

Table 6: Denotational event structure semantics of πIK

$$\left\{\!\left|\overline{a}(n)[m].P\right|\!\right\}_{(\mathcal{N},l)} = \left\langle \overline{a}(n)(e).\mathcal{E}_P, \mathsf{Init}_P \cup \{e\}, k_P[e \mapsto m] \right\rangle \text{ for some fresh } e \notin E \text{ where } \{\!|P|\!\}_{(\mathcal{N},l)} = \langle \mathcal{E}_P, \mathsf{Init}_P, k_P \rangle \text{ and}$$

$$\{\!|a(b)[m].P|\!\}_{(\mathcal{N},l)} = \left\langle \sum_{n\in(\mathcal{N}\setminus\mathsf{sbn}(P))} a(n)(e_n).\mathcal{E}_{P_n}, (\bigcup_{n\in(\mathcal{N}\setminus\mathsf{sbn}(P))} \{n\}\times\mathsf{Init}_{P_n}) \cup \{(b,e_b)\}, k \right\rangle$$
for some fresh $e_n \notin E_n$ where $\{\!|P[b_{[m]} := n]|\!\}_{(\mathcal{N},l)} = \left\langle \mathcal{E}_{P_n}, \mathsf{Init}_{P_n}, k_{P_n} \right\rangle$
$$k((n,e)) = \begin{cases} m & \text{if } e = e_b \text{ and } n = b \\ k_{P_n}(e) & \text{otherwise} \end{cases}$$

$$\{\!|(\nu a)P|\!\}_{(\mathcal{N},l)} = \langle \mathcal{E} \upharpoonright E_\alpha, \mathsf{Init} \cap E_\alpha, k \upharpoonright E_\alpha \rangle \text{ where:}$$
$\{\!|P|\!\}_{(\mathcal{N},l)} = \langle \mathcal{E}, \mathsf{Init}, k\rangle \qquad E_\alpha = \rho(\{\alpha \mid a \in n(\alpha)\})$
if whenever there exist past actions $b(a)[m]$ and $\overline{b}(a)[m]$ in P then they are guarded by a restriction (νa) in P

$$\{\!|(\nu a)P|\!\}_{(\mathcal{N},l)} = \langle \mathcal{E}, \mathsf{Init}, k\rangle \text{ where } \{\!|P|\!\}_{(\mathcal{N},l)} = \langle \mathcal{E}, \mathsf{Init}, k\rangle \text{ and}$$
if there exist past actions $b(a)[m]$ and $\overline{b}(a)[m]$ in P which are not guarded by a restriction (νa) in P

$$\{\!|P_0|P_1|\!\}_{(\mathcal{N},l)} = \left\langle (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act}) \upharpoonright \{e \mid \lambda(e) \neq 0\}, \mathsf{Init}, k \right\rangle \text{ where}$$
for $i \in \{0,1\}$, $\{\!|P_i|\!\}_l = \langle \mathcal{E}_i, \mathsf{Init}_i, k_i\rangle$
$(E_0, F_0, \mapsto_0, \sharp_0, \rhd_0) \times (E_0, F_0, \mapsto_0, \sharp_0, \rhd_0) = (E, F, \mapsto', \sharp, \rhd')$
$$\lambda(e) = \begin{cases} \lambda_0(e_0) & \text{if } e = (e_0, *) \\ \lambda_1(e_1) & \text{if } e = (*, e_1) \\ \tau & \text{if } e = (e_0, e_1) \text{ and} \\ & \text{for } i \in \{0,1\},\ \lambda_i(e_i) = a(x) \text{ and } \lambda_{1-i}(e_{1-i}) = \overline{a}(x) \\ 0 & \text{otherwise} \end{cases}$$
$\mathsf{Act} = \{\tau\} \cup \mathsf{Act}_0 \cup \mathsf{Act}_1$
$\mathsf{Init} = \{(e_0, *) | e_0 \in \mathsf{Init}_0 \text{ and } \nexists e_1 \in \mathsf{Init}_1 . k_1(e_1) = k_0(e_0)\} \cup$
$\{(*, e_1) | e_1 \in \mathsf{Init}_1 \text{ and } \nexists e_0 \in \mathsf{Init}_0 . k_1(e_1) = k_0(e_0)\} \cup$
$\{(e_0, e_1) | e_0 \in \mathsf{Init}_0 \text{ and } e_1 \in \mathsf{Init}_1 \text{ and } k_1(e_1) = k_0(e_0)\}$
$X \mapsto e$ if $X \mapsto' e$ or there exists $x \in no(\lambda(e))$ such that $X = \{e' \mid \exists a . \lambda(e') = \overline{a}(x)\}$ and $x \in \mathsf{sbn}(P)$
$e \rhd \underline{e'}$ if either $e \rhd' \underline{e'}$ or there exists $x \in no(\lambda(e))$ such that $\exists a . \lambda(e') = \overline{a}(x)$
$$k(e) = \begin{cases} k_0(e_0) & \text{if } e = (e_0, *) \text{ or } e = (e_0, e_1) \\ k_1(e_1) & \text{if } e = (*, e_1) \end{cases}$$

$$\{\!|!_{\mathbf{x}} P|\!\}_{(\mathcal{N},0)} = \langle (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset), \emptyset, \emptyset \rangle$$

$$\{\!|!_{\mathbf{x}} P|\!\}_{(\mathcal{N},l)} = \left\{\!\left|!_{\mathbf{x}\setminus\{x_0,\ldots,x_k\}} P | P\{^{x_0,\ldots,x_k}/_{a_0,\ldots,a_k}\}\right|\!\right\}_{(\mathcal{N},l-1)}$$
if $\{x_0, \ldots, x_k\} \subseteq \mathbf{x}$ and $\mathsf{bn}(P) = \{a_0, \ldots, a_k\}$

Table 6: Denotational event structure semantics of πIK (continued)

Example 6.2. *Consider the process* $a(b)[n] \mid \overline{a}(b)[n]$. *Our event structure semantics*

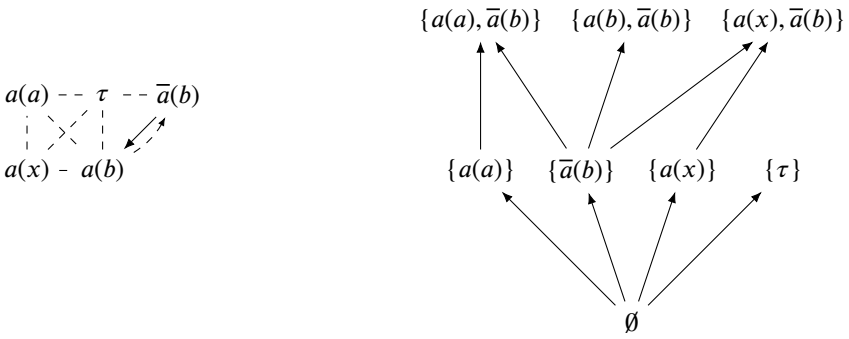

Figure 3: The LRBES seen in Example 6.2 and its configurations.

generate a LRBES $\left\{\!\left\{ a(x)[n] \mid \overline{a}(b)[n] \right\}\!\right\}_{\{a,b,x\}} = \left\langle (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act}), \mathsf{Init}, k \right\rangle$ *where:*

$$
\begin{array}{ll}
E = F = \{a(b), a(a), a(x), \overline{a}(b), \tau\} & \lambda(e) = \ e \\
\{\overline{a}(b)\} \mapsto a(b) & \mathsf{Act} = \ \{a(b), a(a), a(x), \overline{a}(b), \tau\} \\
a(b) \sharp a(a),\ a(b) \sharp a(x),\ a(a) \sharp a(x), & \mathsf{Init} = \ \{\tau\} \\
a(b) \sharp \tau,\ a(a) \sharp \tau,\ a(x) \sharp \tau,\ \overline{a}(b) \sharp \tau & k(\tau) = \ n \\
a(b) \rhd \overline{a}(b) &
\end{array}
$$

From this we see that (1) receiving b is causally dependent on sending b, (2) all the possible inputs on a are in conflict with one another, (3) the synchronisation between the input and the output is in conflict with either happening on their own, and (4) since the two past actions have the same key, the initial state contains their synchronisation.

We show that there exists a least upper bound of our unfolding in Proposition 6.3.

Proposition 6.3 (Unfolding). *Given a process P, a set of names* $\mathcal{N} \supseteq n(P)$*, and a level of unfolding l, if* $\{\!|P|\!\}_{(\mathcal{N},l)} = \langle \mathcal{E}, \mathsf{Init}, k \rangle$ *and* $\{\!|P|\!\}_{(\mathcal{N},l-1)} = \left\langle \mathcal{E}', \mathsf{Init}', k' \right\rangle$*, then* $\mathcal{E}' \leq \mathcal{E}$*,* $\mathsf{Init} = \mathsf{Init}'$*, and* $k = k'$*.*

PROOF. Follows from Lemmas E.1 to E.4 in Appendix E.1, and $\mathcal{E} \times (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset) \cong \mathcal{E}$.

Proposition 6.4. *Let P be a forwards reachable process wherein all bound and free names are different and* $\mathcal{N} \supseteq n(P)$ *be a set of names. If* $\{\!|P|\!\}_{\mathcal{N}} = \langle \mathcal{E}, \mathsf{Init}, k \rangle$ *then* $\mathcal{E}$ *is causal.*

PROOF. Follows from the event structure semantics.

Proposition 6.5 shows that structurally congruent processes generate isomorphic event structures.

Proposition 6.5 (Structural Congruence). *Given two processes P and* P' *and a set of names* $\mathcal{N} \supseteq n(P) \cup n(P')$*, if* $P \equiv P'$*,* $\{\!|P|\!\}_{\mathcal{N}} = \langle \mathcal{E}, \mathsf{Init}, k \rangle$*, and* $\{\!|P'|\!\}_{\mathcal{N}} = \left\langle \mathcal{E}', \mathsf{Init}', k' \right\rangle$*, then there exists an isomorphism* $f : \mathcal{E} \to \mathcal{E}'$ *such that* $f(\mathsf{Init}) = \mathsf{Init}'$ *and for all* $e \in \mathsf{Init}$*,* $k(e) = k'(f(e))$*.*

PROOF. We prove this by case analysis on the structural congruence rules in Appendix E.2

We show in Theorems 6.7 and 6.8 that given a process P with a conflict-free initial state, including any reachable process, there exists a transition $P \xrightarrow{\mu[m]} P'$ if and only if the event structure corresponding to P is isomorphic to the event structure corresponding to P' and an event labelled μ is available in P's initial state, and P''s initial state is P's initial state with this event added.

Lemma 6.6. *If P is a forwards reachable process and $\mathcal{N} \supseteq n(P)$ is a set of names with $\{\!|P|\!\}_{\mathcal{N}} = \langle \mathcal{E}, \mathsf{Init}, k \rangle$. Then $\mathsf{Init} = \emptyset$ if and only if $\mathsf{std}(P)$.*

PROOF. The only rules in the event structure semantics adding events to an empty initial set involve past actions, and the only rule removing events from the initial state requires the event to be behind a restriction not created by a communication.

Theorem 6.7. *Let P be a forwards reachable process wherein all bound and free names are different and let $\mathcal{N} \supseteq n(P)$ be a set of names. If (1) $\{\!|P|\!\}_{\mathcal{N}} = \langle \mathcal{E}, \mathsf{Init}, k \rangle$ where $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$, and Init is conflict-free, and (2) there exists a transition $P \xrightarrow{\mu[m]} P'$ such that $\{\!|P'|\!\}_{\mathcal{N}} = \left\langle \mathcal{E}', \mathsf{Init}', k' \right\rangle$, then there exists an isomorphism $f : \mathcal{E} \to \mathcal{E}'$ and a transition in $C_{br}(\mathcal{E})$, $\mathsf{Init} \xrightarrow{\{e\}} X$, such that $\lambda(e) = \mu$, $f \circ k' = k[e \mapsto m]$, and $f(X) = \mathsf{Init}'$.*

PROOF. We prove this by induction on $P \xrightarrow{\mu[m]} P'$ in Appendix E.3.

Theorem 6.8. *Let P be a forwards reachable process wherein all bound and free names are different and let $\mathcal{N} \supseteq n(P)$ be a set of names. If (1) $\{\!|P|\!\}_{\mathcal{N}} = \langle \mathcal{E}, \mathsf{Init}, k \rangle$ where $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$, and (2) there exists a transition $\mathsf{Init} \xrightarrow{\{e\}} X$ in $C_{br}(\mathcal{E})$, then there exists a transition $P \xrightarrow{\mu[m]} P'$ such that $\{\!|P'|\!\}_{\mathcal{N}} = \left\langle \mathcal{E}', \mathsf{Init}', k' \right\rangle$ and an isomorphism $f : \mathcal{E} \to \mathcal{E}'$ such that $\lambda(e) = \mu$, $f \circ k' = k[e \mapsto m]$, and $f(X) = \mathsf{Init}'$.*

PROOF. We prove this by structural induction on P in Appendix E.4.

Corollary 6.9. *Let P be a forwards reachable process and $\mathcal{N} \supseteq n(P)$ be a set of names such that $\{\!|P|\!\}_{\mathcal{N}} = \langle \mathcal{E}, \mathsf{Init}, k \rangle$. Then Init is forwards-reachable in $\mathcal{E}$ if and only if there exists a standard process Q such that $Q \to^* P$.*

PROOF. Follows from Lemma 6.6 and Theorems 6.8 and 6.7.

We then prove the same operational correspondence on reverse transitions, with the event being removed rather than added to the initial state.

Theorem 6.10. *Given a process P, if $\{\!|P|\!\} = \langle \mathcal{E}, \mathsf{Init}, k \rangle$, $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$, $C_{br}(\mathcal{E}) = (E, F, \mathsf{C}, \to)$, Init is conflict-free, and there exists a transition $P \overset{\mu[m]}{\rightsquigarrow} P'$ such that $\{\!|P'|\!\} = \left\langle \mathcal{E}', \mathsf{Init}', k' \right\rangle$, then there exist isomorphisms $f : \mathcal{E} \to \mathcal{E}'$ and $g : \mathcal{E}' \to \mathcal{E}$ and a transition $\mathsf{Init} \xrightarrow{\{\underline{e}\}} X$ such that $\lambda(e) = \mu$, $f \circ k' = k[e \mapsto m]$, and $f(X) = \mathsf{Init}'$.*

PROOF. Implied by Proposition 6.4, Theorem 6.7, and Corollary 6.9.

Theorem 6.11. *Given a process P, if $\{\!|P|\!\} = \langle \mathcal{E}, \mathsf{Init}, k\rangle$, $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$, $C_{br}(\mathcal{E}) = (E, F, \mathsf{C}, \rightarrow)$, Init is conflict-free, and there exists a transition $\mathsf{Init} \xrightarrow{e} X$ in $C_{br}(\mathcal{E})$, then there exists a a key m and a transition $P \overset{\lambda(e)[m]}{\rightsquigarrow} P'$, such that $\{\!|P'|\!\} = \left\langle \mathcal{E}', \mathsf{Init}', k' \right\rangle$ and there exist isomorphisms $f : \mathcal{E} \to \mathcal{E}'$ and $g : \mathcal{E}' \to \mathcal{E}$ such that $f \circ k' = k[e \mapsto m]$ and $f(X) = \mathsf{Init}'$.*

PROOF. Implied by Proposition 6.4, Theorem 6.8, and Corollary 6.9.

7. Operational event structure semantics of πIH-calculus

In this section, we define event structure semantics of the πIH calculus. Unlike the denotational event structure semantics we have previously defined, these operational semantics generate event structures from the transitions rather than the syntax.

We generate a labelled asynchronous transition system (LATS) (Definition 7.1) based on the causal semantics of πIH.

Definition 7.1 (Labelled asynchronous transition system [32]). A labelled asynchronous transition system (LATS) is a tuple $(S, i, E, \mathcal{T}, I, \mathsf{lab}, \mathcal{A})$, where

- $(S, i, E, \mathcal{T})$ is a transition system with the set of states S, the initial state i, the set of events E, and the transition relation $\mathcal{T} \subseteq S \times E \times S$;
- $\mathsf{lab} : E \to \mathcal{A}$ is the labelling function from the set of events to the set of actions;
- $I \subseteq E \times E$ is the irreflexive, symmetric independence relation satisfying:
  - $e \in E \Rightarrow \exists s, s' \in S.(s, e, s') \in \mathcal{T}$;
  - $(s, e, s') \in \mathcal{T} \wedge (s, e, s'') \in \mathcal{T} \Rightarrow s' = s''$;
  - $e_0 I e_1 \wedge \{(s, e_0, s_0), (s, e_1, s_1)\} \subseteq \mathcal{T} \Rightarrow \exists s_2.\{(s_0, e_1, s_2), (s_1, e_0, s_2)\} \subseteq \mathcal{T}$;
  - $e_0 I e_1 \wedge \{(s, e_0, s_0), (s_0, e_1, s_2)\} \subseteq \mathcal{T} \Rightarrow \exists s_1.\{(s, e_1, s_1), (s_1, e_0, s_2)\} \subseteq \mathcal{T}$.

Definition 7.2 (Generated transition system [14]). A πIH process generates a transition system $TS = (S, i, E, \mathcal{T}, I, \mathsf{lab}, \mathcal{A})$ defined as follows:

- The set of states, S contains equivalence classes with respect to $\equiv$ as defined in Section 2 of pairs of histories and processes:
  $S = \left\{ \mathbb{P} \;\middle|\; \forall \mathbf{H} \vdash P, \mathbf{H}' \vdash P' \in \mathbb{P} : \mathbf{H} = \mathbf{H}' \text{ and } P \equiv P' \right\}$
- the set of events, E, consists of triples of actions, locations, and causes:
  $E = \left\{ (\alpha, u, D) \;\middle|\; \exists \mathbf{H}, P, \mathbf{H}', P' : \mathbf{H} \vdash P \xrightarrow[u,D]{\alpha} \mathbf{H}' \vdash P' \right\}$
  which are labelled with their action, α
- $\mathcal{T} \subseteq S \times E \times S$ are the transitions from Definition 2.7.
  $\mathcal{T} = \left\{ \mathbb{P}, (\alpha, u, D), \mathbb{P}') \;\middle|\; \exists \mathbf{H} \vdash P \in \mathbb{P}, \mathbf{H}' \vdash P' \in \mathbb{P}' : \; \mathbf{H} \vdash P \xrightarrow[u,D]{\alpha} \mathbf{H} \vdash P' \right\}$

- Events (α_0, u_0, D_0) and (α_1, u_1, D_1) are independent if $\overset{\alpha_0}{\underset{u_0,D_0}{\rightarrowtail}}$ and $\overset{\alpha_1}{\underset{u_1,D_1}{\rightarrowtail}}$ are independent according to Definition 2.9.

We use action, location, and causes respectively to extract the action, location, and causes from events.

For a particular process P, we select the initial state $i = \{(\emptyset, \emptyset, \emptyset) \vdash P\}_{\equiv}$, and restrict the transition system to the part reachable from this state to get $TS(P)$.

Theorem 7.3. *TS is an LATS.*

PROOF. See Appendix F.1.

In Definition 7.2, we showed how to generate an LATS using our extrusion history-based semantics. In [32], a mapping from LATSs to labelled prime event structures (LPESs) via trace languages was defined, which we can use this to generate event structures from πIH processes. An example of this applied to a πIH-calculus process can be seen in Example 7.5. Since events are sets of traces, independent actions with conflicting causes become conflicting events.

Definition 7.4 (From LATS to LPES [32]). Given LATS $\mathcal{T} = (S, i, E, \mathcal{T}, I, \mathsf{lab}, \mathcal{A})$, we can generate a labelled prime event structure $LP(\mathcal{T}) = (E', <, \sharp, \lambda, \mathcal{A})$ where:

- E' is the equivalence classes of non-empty forward-only traces of $\mathcal{T}$ with respect to $\approx$, where $\approx$ is the smallest equivalence relation on non-empty traces such that $ta \approx tba$ if bIa and $sa \approx ta$ if $s\overline{\sim}t$

  where $\overline{\sim}$ is the smallest equivalence relation such that $sabt\overline{\sim}sbat$ if aIb

- Given $e, e' \in E'$, $e < e'$ if there exists a trace $s \in e$ such that for all $t \in e'$, there exists $t'\overline{\sim}t$ such that s is a prefix of t'.

- Given $e, e' \in E'$, $e \sharp e'$ if there exist $e_0 \leq e$ and $e'_0 \leq e'$ such that there exist traces $t_0 a \in e_0$ and $t_0 b \in e'_0$ such that $a \neq b$ and $\neg(aIb)$

- For $e \in E'$, if $t.a \in e$ then $\lambda(e) = \mathsf{lab}(a)$.

Example 7.5. *Consider the process $(\overline{\emptyset}, \underline{\emptyset}, \emptyset) \vdash a(x)|\overline{a}(b)$. This generates the LATS (where we leave out traces using α-conversion and only consider one possible free name as input, as otherwise there would infinitely many events and states) seen in Figure 4 with the independence relation $(a(x), 0[a(x)][0], \emptyset)I(\overline{a}(b), 1[\overline{a}(b)][0], \emptyset)$.*

This gives us the events:

1. $\{(a(x), 0[a(x)][0], \emptyset);\ (\overline{a}(b), 1[\overline{a}(b)][0], \emptyset), (a(x), 0[a(x)][0], \emptyset)\}$
2. $\{(\overline{a}(b), 1[\overline{a}(b)][0], \emptyset);\ (a(x), 0[a(x)][0], \emptyset), (\overline{a}(b), 1[\overline{a}(b)][0], \emptyset)\}$
3. $\{(\tau, \left\langle 0[a(x)][0], 1[\overline{a}(b)][0] \right\rangle, \emptyset)\}$
4. *and* $\{(\overline{a}(b), 1[\overline{a}(b)][0], \emptyset), (a(b), 0[a(x)][0], \{(\overline{a}(b), 1[\overline{a}(b)][0])\})\}$

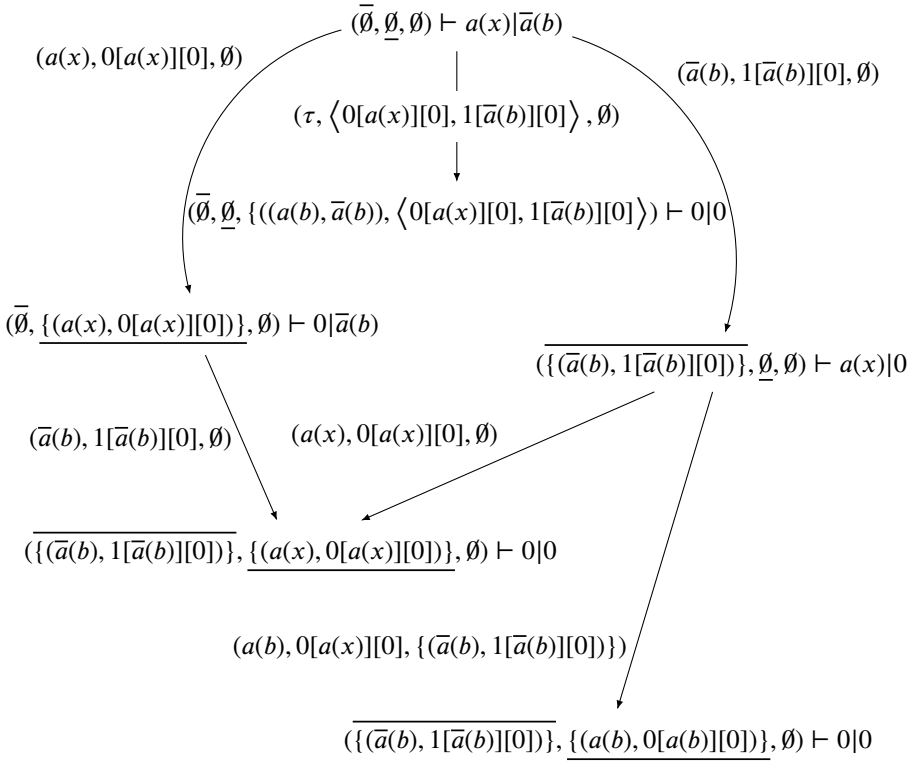

Figure 4: The LATS generated by the process in Example 7.5

with $2 < 4$, $1 \sharp 3$, *and* $2 \sharp 3$.

Because 4 *is the only event which does not contain a trace of length 1, it is also the only event with a cause. Events* 1 *and* 2 *are in conflict with* 3 *because* $0[a(x)][0]$ *and* $1[\overline{a}(b)][0]$ *are not independent of* $\langle 0[a(x)][0], 1[\overline{a}(b)][0] \rangle$.

In the denotational event structure semantics 1 *would be an event labelled* $a(x)$, 2 *would be an event labelled* $\overline{a}(b)$, 3 *would be an event labelled* τ, *and* 4 *would be an event labelled* $a(b)$.

Finally, since we are modelling a reversible calculus, we need to turn our LPES into a causal labelled reversible prime event structure (CLRPES).

Definition 7.6 (Causal Labelled Reversible Prime Event Structure [11]). A *labelled reversible prime event structure* (LRPES) is a sextuple $\mathcal{E} = (E, F, <, \sharp, \prec, \rhd, \lambda, \mathsf{Act})$ where E is the set of events and

1. $<$ is an irreflexive partial order such that for every $e \in E$, $\{e' \in E \mid e' < e\}$ is finite and conflict-free;
2. $\sharp$ is irreflexive and symmetric;
3. if $e < e'$ then not $e \sharp e'$;

4. $F \subseteq E$ is the set of reversible events;
5. $\rhd \subseteq E \times \underline{F}$ is the prevention relation;
6. $\prec \subseteq E \times \underline{F}$ is the reverse causality relation where for each $e \in F$, $e \prec \underline{e}$ and $\{e' \mid e' \prec \underline{e}\}$ is finite and conflict-free;
7. if $e \prec \underline{e'}$ then not $e \rhd \underline{e'}$;
8. $\sharp$ is hereditary with respect to *sustained causation* $\prec$, where $e \prec e'$ means that $e < e'$ and if $e \in F$ then $e' \rhd \underline{e}$;
9. $\prec$ is transitive;
10. $\lambda : E \to \mathsf{Act}$ is a labelling function.

A *causal* LRPES (CLRPES) $\mathcal{E} = (E, F, <, \sharp, \prec, \rhd, \lambda, \mathsf{Act})$ is an RPES such that for all $e \in E$ and $e' \in F$, $e' < e$ if and only if $e \rhd \underline{e'}$, and $e \prec \underline{e'}$ if and only if $e = e'$.

Definition 7.7 (From LPES to CLRPES [11]). Given a PES, $(E, <, \sharp, \lambda, \mathsf{Act})$, generated from a process, $LR((E, <, \sharp, \lambda, \mathsf{Act})) = (E, F, <, \sharp, \prec, \rhd, \lambda, \mathsf{Act})$ where $F = E$; $e' \prec \underline{e'}$ for all $e' \in F$; and $e \rhd \underline{e'}$ if and only if $e' < e$.

8. Correspondence between denotational and operational event structure semantics

With event structure semantics defined for two variants of reversible πI-calculus, we will now compare them. Example 8.1 shows the difference between how our processes are represented by LRBESs and LRPESs, and illustrates the difficulty of mapping LRPESs to LRBESs in a way, that gives us the intended result. We therefore create a function to turn the CLRBESs into CLRPESs (Definition 8.3). We do this by splitting each event into multiple events, based on the different sets of *possible causes*.

Example 8.1 (Turning a LRPES into a LRBES). *The LRPES generated by the process* $(\emptyset, \emptyset, \emptyset) \vdash \overline{a}(b).b(x)|a(y)$ *has two events labelled* $b(x)$*, one which is caused by an event labelled* $\overline{a}(b)$*, and another caused by an event labelled* τ*, representing the communication on a. The obvious way to represent this in an LRBES would be to keep two events labelled* $b(x)$ *and say* $\{\overline{a}(b)\} \mapsto b(x)_1$ *and* $\{\tau\} \mapsto b(x)_2$*.*

However, if we look at $\left\{\!\left|\overline{a}(b).b(x)|a(y)\right|\!\right\}_{\{a,b,x,y\}}$*, we get one event labelled* $b(x)$*, with* $\{\overline{a}(b), \tau\} \mapsto b(x)$*.*

These RBESs have the same configurations and transitions (up to labels), but are not isomorphic, and determining which events and bundles need to be merged from a larger RPES would be difficult.

Recall that we defined the possible causes of an event in a CRBES in Definition 5.3. We now give an alternative equivalent definition in Lemma 8.2. Requiring that X is finite and $e \notin X$ for any RPES event (X, e) means we give each event a finite set of causes, and the event does not cause itself. Some processes can generate cycles of bundles, such as $a(x).\overline{b}(c)|d(y).\overline{e}(f)$, where $a(f)$ is a possible cause of $\overline{b}(c)$, which is a possible cause of $d(c)$, which is a possible cause of $\overline{e}(f)$, which is a possible cause of $a(f)$. These kinds of cycles are allowed in LRBESs, but when transforming them into LRPESs, we only take the instance of $a(f)$, which was not transitively caused by $d(c)$, but instead by d receiving some other name.

Lemma 8.2. *Given a CLRBES, $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$, $X \subseteq E$ is a possible cause of $e \in E$ if and only if in $C_{br}(\mathcal{E})$,*

1. *X is a reachable and finite configuration;*
2. *$X \xrightarrow{e}$;*
3. *for all reachable configurations $X'' \subset X$, $X'' \not\xrightarrow{e}$.*

PROOF. We prove that if X is a possible cause, then it fulfils these conditions by induction on the size of X. We the prove that if X fulfils these conditions then it fits Definition 5.3. The full proof can be seen in Appendix G.1.

Having defined possible causes, we can use them to define a mapping from the CLRBESs generated by our denotational semantics to LRPESs which can more easily be compared with the LRPESs generated by our operational semantics.

Definition 8.3 (CLRBES to LRPES). Given a CLRBES $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$, we can create an LRPES $p_B(\mathcal{E}) = (E', F', <, \sharp', \prec, \rhd', \lambda', \mathsf{Act}) \upharpoonright \{e \mid e' < e \text{ and } e'' < e \Rightarrow e' \sharp e''\}$ where:

1. $E' = \{(X, e) \mid X \text{ is a possible cause of } e\}$;
2. $F' = \{(X, e) \mid (X, e) \in E' \text{ and } e \in F\}$;
3. $(X, e) < (X', e')$ if $X \cup \{e\} \subseteq X'$;
4. $(X, e)\ \sharp''\ (X', e')$ if $e \sharp e'$ or both $e = e'$ and $X \neq X'$, and $\sharp'$ is $\sharp''$ closed under conflict heredity and symmetry;
5. $\prec = \emptyset \times \emptyset$;
6. $(X, e) \rhd' \underline{(X, e')}$ if $(X', e') < (X, e)$;
7. $\lambda'((X, e)) = \lambda(e)$.

Definition 8.4. We define a functor P_B such that:

- $P_B(\mathcal{E}) = p_B(\mathcal{E})$;
- $P_B(f) = f'$ where $f'((X, e)) = (f(X), f(e))$.

Proposition 8.5. *P_B is a functor.*

PROOF. See Appendix G.2 for full proof.

Lemma 8.6 (P_B preserves transitions). *Given a CLRBES, $\mathcal{E}$, and a reachable configuration, Y, of $C_{br}(\mathcal{E})$, $Y \xrightarrow{e}$ if and only if there exists a reachable configuration of $C_{pr}(P_B(\mathcal{E}))$, $Y' = \{(X', e') \mid X' \cup \{e'\} \subseteq Y\}$ and $X \subseteq Y$ such that $Y' \xrightarrow{(X,e)}$.*

PROOF. We prove this by induction on the size of Y. The full proof can be seen in Appendix G.3.

Now that both our event structure semantics can generate LRBESs, we can show that the LRPES generated by the denotational semantics is isomorphic with the sub-LRPES of the LRPES generated by the operational semantics, which does not include any actions generated by α-converting apart from when unfolding replication.

In Example A.1 in Appendix A we show the event structures generated by the two different semantics for the process $a(x)|\overline{a}(b).\overline{b}(c)$. We see that we can create morphisms between the two LRPESs if we restrict the event structure generated by the operational semantics to actions involving the names considered by the denotational semantics.

We now show isomorphism between the two event structure semantics.

Theorem 8.7. *Given a standard πI-calculus process P with guarded choice, distinct names and $\{\!|P|\!\}_{\mathcal{N}} = \langle \mathcal{E}, \mathsf{Init}, k\rangle$ for some $\mathcal{N} \supseteq n(P)$ and $\mathcal{E}'$ being the event structure generated by the causal semantics of $(\overline{\emptyset}, \emptyset, \emptyset) \vdash P$, we can define two LRPES morphisms, $f : P_B(\mathcal{E}) \to \mathcal{E}'$ and $f' : \mathcal{E}' \to P_B(\mathcal{E})$, such that $f \circ f' = 1_{P_B(\mathcal{E})}$ and if $f'(e')$ is defined, then $f(f'(e')) = e'$.*

PROOF. We prove this by induction on the level of nested replication and structure of P in Appendix G.4.

The operational event structure semantics generate additional events corresponding to the actions the process can take if α-converted, and the input actions receiving names not in $\mathcal{N}$. We would like to show that the way these events interact corresponds to the way the events generated by the denotational semantics interact, although the events will have different labels. In order to get a correspondence between events generated by α-converting the process and events generated without α-converting the process, we wish to define substitution on traces.

We specifically want to define substitution of a name x for a in trace t, at location u in which a is bound, but not continuing after a set of locations, U, at which new xs are bound. To do this we look at each triple (μ, u', D) in t in order, and determine whether it is in a location where the substitution should take place, that is u' is within the gap from u to U as defined in Definition 8.8. If this is the case, then we substitute free occurrences of x for a in μ and the processes of u'. If x is bound in μ, then we add u' to U before continuing. If a is bound in μ then we substitute a for some new name, d. If μ is an input of x and u' is not a location where the substitution should take place, then we use D to determine whether the x being received is the same as the one being substituted or a different one. If μ is τ, then we use the processes in u' to figure out which channel communicated which name, and then treat it as separate input and output. The full definition can be seen in Definition 8.9.

Definition 8.8 (location order). We define an ordering on locations, $\prec$, as follows:

- $l[P][Q] \prec l'[P'][Q']$ if $l' = ll''$ or $l' = l$ and $Q \to^* P'$;
- $l\langle l_0[P_0][Q_0], l_1[P_1][Q_1]\rangle \prec l'[P'][Q']$ if there exists an $i \in \{0,1\}$ such that $ll_i[P_i][Q_i] \prec l'[P'][Q']$;
- $l[P][Q] \prec l'\left\langle l'_0[P'_0][Q'_0], l'_1[P'_1][Q'_1]\right\rangle$ if there exists an $i \in \{0,1\}$such that $l[P][Q] \prec l'l'_i[P'_i][Q'_i]$;
- $l\langle l_0[P_0][Q_0], l_1[P_1][Q_1]\rangle \prec l'\left\langle l'_0[P'_0][Q'_0], l'_1[P'_1][Q'_1]\right\rangle$ if for some $i, j \in \{0,1\}$, we have $ll_i[P_i][Q_i] \prec l'l'_j[P'_j][Q'_j]$.

Given a start location u and a set of end locations U, we say that u' is within the gap from u to U, $\mathsf{within}_{u-U}(u')$, if $u \prec u'$ and there does not exist $u'' \in U$ such that $u'' \prec u'$.

Definition 8.9. Given a trace t, a location u, a set of locations, U, and names a, x, t with x substituted for a starting at location u but not continuing after locations in U, $t[x := a]_{(u-U)}$ is defined as:

1. $((\alpha, l[P][Q], D)t)[x := a]_{(u-U)} = (\alpha', l[P'][Q'], D')t'$ where:
   (a) $\alpha' = \alpha''[x := a]$ if $\mathsf{within}_{u-U}(l[P][Q])$ and otherwise $\alpha' = \alpha''$
   where if $\alpha = \overline{x}(n)$, $\alpha[x := a] = \overline{a}(n)$ and if $\alpha = x(n)$, $\alpha[x := a] = a(n)$ and otherwise $\alpha[x := a] = \alpha$.
   And $\alpha'' = b(a)$ if $\alpha = b(x)$ and $(x, u') \in D$, $\alpha'' = b(d)$ if $\alpha = b(a)$, $\alpha'' = \overline{b}(d)$ if $\alpha = \overline{b}(a)$, and otherwise $\alpha'' = \alpha$.
   (b) $P' = P[x := a]$ if $\mathsf{within}_{u-U}(l[P][Q])$ and otherwise $P' = P$.
   (c) $Q' = Q''[x := a]$ if $\mathsf{within}_{u-U}(l[P][Q])$ and otherwise $Q' = Q''$
   where $Q'' = Q[a := d]$ if $a \in \mathsf{bn}(\alpha)$ and $\mathsf{within}_{u-U}(l[P][Q])$, and otherwise $Q'' = Q$.
   (d) $D' = D''[x := a]_{(u-U)}$ where $D'' = D[a := d]_{(l[P][Q]-\emptyset)}$ if $a \in \mathsf{bn}(\alpha)$ and $\mathsf{within}_{u-U}(l[P][Q])$, and otherwise $D'' = D$.
   (e) $t' = t''[x := a]_{(u-U')}$ where $U' = U \cup \{l[P'][Q']\}$ if $x \in \mathsf{bn}(\alpha)$ and otherwise $U' = U$
   where $t'' = t[a := d]_{(l[P][Q]-\emptyset)}$ if $a \in \mathsf{bn}(\alpha)$ and $\mathsf{within}_{u-U}(l[P][Q])$, $t'' = t[x := a]_{(u-\emptyset)}$ if $\alpha = b(x)$, and otherwise $t'' = t$.
   for $d \notin \mathsf{fn}(Q)$
2. $((\tau, l\,\langle l_0[P_0][Q_0], l_1[P_1][Q_1]\rangle, D)t)[x := a]_{(u-U)} =$
   $((\tau, l\,\langle l_0[P_0'][Q_0'], l_1[P_1'][Q_1']\rangle, D')t')$
   where $P_i \xrightarrow{b(n)} Q_i$ and $P_{1-i} \xrightarrow{\overline{b}(n)} Q_{1-i}$ and:
   (a) $P_j' = P_j[x := a]$ if $\mathsf{within}_{u-U}(ll_j[P_j][Q_j])$
   (b) $Q_j' = Q_j''[x := a]$ if $\mathsf{within}_{u-U}(ll_j[P_j][Q_j])$ and $n \neq x$, and otherwise $Q_j' = Q_j''$
   where $Q_j'' = Q_j[a := d]$ if $n = a$ and $\mathsf{within}_{u-U}(l\,\langle l_0[P_0][Q_0], l_1[P_1][Q_1]\rangle)$, and otherwise $Q_j'' = Q_j$.
   (c) $D' = D''[x := a]_{(u-U)}$ where $D'' = D[a := d]_{(l\langle l_0[P_0][Q_0], l_1[P_1][Q_1]\rangle-\emptyset)}$ if both $n = a$ and $\mathsf{within}_{u-U}(l\,\langle l_0[P_0][Q_0], l_1[P_1][Q_1]\rangle)$, and otherwise $D'' = D$
   (d) $t' = t''[x := a]_{(u-U')}$ where $t'' = t[a := d]_{(l\langle l_0[P_0][Q_0], l_1[P_1][Q_1]\rangle-\emptyset)}$ if both $n = a$ and $\mathsf{within}_{u-U}(l\,\langle l_0[P_0][Q_0], l_1[P_1][Q_1]\rangle)$, and otherwise $t'' = t$
   where $U' = U \cup \{l\,\langle l_0[P_0][Q_0], l_1[P_1][Q_1]\rangle\}$ if $n = x$ and otherwise $U' = U$.
   for $d \notin \mathsf{fn}(Q_0) \cup \mathsf{fn}(Q_1)$

Here we use the following definition of substitution on a set of causes:

$D[x := a]_{(u-U)} = \{(\alpha'', u'') \mid (\alpha', l[P][Q]) \in D$ and if $l[P][Q] \prec u$ or there exists $u' \in U$ such that $u' \prec l[P][Q]$, then $\alpha'' = \alpha'$ and $u'' = l[P][Q]$, otherwise $\alpha'' = \alpha'[x := a]$ and $u'' = l[P[x := a]][Q[x := a]]\}$

We then use this definition to prove that we can "α-convert" traces by substituting a bound name in an input or output action in the trace and then substituting that name on that location in the rest of the trace.

Lemma 8.10. *Given a πIH-calculus process with empty extrusion histories, $(\emptyset, \emptyset, \emptyset) \vdash P$, and a trace, $t = (\mu_0, u_0, D_0), \ldots, (\mu_n, u_n, D_n)$ such that*

$$(\emptyset, \emptyset, \emptyset) \vdash P \xrightarrow[u_0, D_0]{\mu_0} \mathbf{H}_1 \vdash P_1 \ldots \mathbf{H}_n \vdash P_n \xrightarrow[u_n, D_n]{\mu_n} \mathbf{H}' \vdash P'$$

Then for any $(\mu_i, l_i[Q_i][Q_i'], D_i)$ such that either $\mu_i = \overline{b}(x)$ or $\mu_i = b(x)$ and, we can define new traces

$$t_0 = (\mu_0, u_0, D_0) \ldots, (\mu_{i-1}, u_{i-1}, D_{i-1})$$

and

$$t_1 = (\mu_{i+1}, u_{i+1}, D_{i+1}), \ldots, (\mu_n, u_n, D_n)[x := a]_{(l_i[Q_i][Q_i']-\emptyset)}$$

and combine them to get

$$t' = t_0(\mu_i', l_i[Q_i][Q_i[x := a]], D_i')t_1 = (\mu_0', u_0', D_0'), \ldots, (\mu_n', u_n', D_n')$$

where $\mu_i' = b(a)$ if $\mu_i = b(x)$ and $\mu_i' = \overline{b}(a)$ if $\mu_i = \overline{b}(x)$, and $D_i' = (D_i \rightharpoonup b) \cup (a, u)$ if $(\overline{c}(a), u \in \overline{H}_i$ and otherwise $D_i' = (D_i \rightharpoonup b)$.

We can then use this trace to get

$$(\emptyset, \emptyset, \emptyset) \vdash P \xrightarrow[u_0', D_0']{\mu_0'} \ldots \xrightarrow[u_n', D_n']{\mu_n'}$$

PROOF. See Appendix G.5.

Finally we show our label-ignoring morphism from all the events generated by the operational semantics to the events generated by the denotational semantics. For this we use a label removing function $\mathsf{lr}((E, F, <, \sharp, \prec, \rhd, \lambda, \mathsf{Act})) = (E, F, <, \sharp, \prec, \rhd)$.

Theorem 8.11. *Given a standard πI-calculus process P with guarded choice, distinct names and $\{\!|P|\!\}_{\mathcal{N}} = \langle \mathcal{E}, \mathsf{Init}, k \rangle$ for some $\mathcal{N} \supseteq n(P)$ and $\mathcal{E}'$ being the event structure generated by the causal semantics of $(\overline{\emptyset}, \overline{\emptyset}, \overline{\emptyset}) \vdash P$, and f and f' defined as above, we have an RPES morphism $f'' : \mathsf{lr}(\mathcal{E}') \to \mathsf{lr}(P_B(\mathcal{E}))$ such that $\mathsf{dom}(f'') = E'$ and $f'' \upharpoonright \mathsf{dom}(f') = f'$.*

PROOF. We prove this by induction on the level of nested replication and structure of P in Appendix G.6 using Lemma 8.10.

9. Conclusion and future work

All existing reversible versions of the π-calculus use reduction semantics [17, 30] or late semantics [8, 21], despite the early semantics being used more widely than the late in the forward-only setting. We have introduced πIH and πIK, the first reversible early π-calculi. They are reversible forms of the *internal* π-calculus, where names being sent in output actions are always bound. As well as structural causation, as in CCS, the early form of the internal π-calculus also has a form of link causation created by the semantics being early, which is not present in other reversible π-calculi. In πIH, past actions are tracked by using extrusion histories adapted from [14], which move past actions and their locations into separate histories for dynamic reversibility. In contrast, πIK keeps the structure of the process intact but annotates past actions with keys, similarly to the framework for reversible π-calculi [21] and CCSK [25] We showed that a process πIH with extrusion histories can be mapped to a πIK process with keys, creating an operational correspondence (Theorem 4.8). This is similar to the correspondence proved between statically reversible CCS (CCSK) and dynamically reversible CCS (RCCS) by [16], however we are not aware of such a mapping between π-calculi.

The event structure semantics of πIK, are defined inductively on the syntax of the process. We use labelled reversible bundle event structures [13], rather than prime event structures, to get a more compact representation where each action in the calculus has only one corresponding event. While causation in the internal π-calculus is simpler that in the full π-calculus, our early semantics means that we still have to handle link causation, in the form of an input receiving a free name being caused by a previous output of that free name. We show an operational correspondence between πIK processes and their event structure representations in Theorems 6.7 and 6.8. Cristescu *et al.* [9] have used rigid families [5], related to event structures, to describe the semantics of Rπ [8]. However, unlike our denotational event structure semantics, their semantics require one to reverse every action in the process before applying the mapping to a rigid family, and then redo every reversed action in the rigid family. Our approach of using a static calculus as an intermediate step means we get the current state of the event structure immediately, and do not need to redo the past steps.

The event structure semantics of πIH on the other hand are defined operationally, being derived from a labelled asynchronous transition system. This creates a LRPES in which each event is a set of traces of the process, all of which end with the same action and contain the same causes of this action.

The denotational semantics assume that the process has already been α-converted so all free and bound names are distinct, and only uses α-conversion when expanding a replication. Therefore the denotational event structure semantics do not include actions requiring α-conversion, and we do not get an exact correspondence between the two event structure semantics, as the events requiring α-conversion in the RPES created by the operational semantics will not have corresponding events with the same labels in the LRPES created by the denotational semantics. However, we do get a morphism mapping all actions requiring α-conversion to an equivalent action in the denotationally generated RPES (Theorem 8.11), which when restricted to only the events present in both event structures forms one half of an isomorphism between the LRPESs generated by the denotational and the operational semantics (Theorem 8.7).

With these morphisms we have shown that not only do we have a correspondence between the statically and dynamically reversible calculi, but between the event structures generated by their operational and denotational event structure semantics. This does not mean there are no benefits to using one calculus over the other. Having a notion of independence of transitions meant we could easily prove useful properties of πIH, which we do not have for πIK. On the other hand, not keeping past actions in a shared memory makes πIK more compositional than πIH. This is also reflected in the event structure semantics, which are more compositional for πIK, but were simpler to define and prove correctness of based on existing mappings for πIH.

Future work:. We could expand the event structure semantics of πIK to a statically reversible full π-calculus. This would entail significantly more link causation, but would give us event structure semantics of a full π-calculus. Another possibility is to expand πIH to get a full reversible early π-calculus.

Acknowledgements:. We thank Thomas Hildebrandt and Håkon Normann for discussions on how to translate their work on π-calculus with extrusion histories to a reversible setting. We thank the anonymous reviewers of RC 2020 for their helpful comments.

This work was partially supported by an EPSRC DTP award; also by the following EPSRC projects: EP/V000462/1, EP/K034413/1, EP/K011715/1, EP/L00058X/1, EP/N027833/1, EP/T006544/1, EP/N028201/1 and EP/T014709/1; and by EU COST Action IC1405 on Reversible Computation.

References

[1] Aubert, C., Cristescu, I.: Contextual equivalences in configuration structures and reversibility. Journal of Logical and Algebraic Methods in Programming **86**(1), 77 – 106 (2017). https://doi.org/10.1016/j.jlamp.2016.08.004

[2] Boreale, M.: On the expressiveness of internal mobility in name-passing calculi. Theoretical Computer Science **195**(2), 205 – 226 (1998). https://doi.org/10.1016/S0304-3975(97)00220-X

[3] Boudol, G., Castellani, I.: Flow models of distributed computations: Three equivalent semantics for CCS. Information and Computation **114**(2), 247 – 314 (1994). https://doi.org/10.1006/inco.1994.1088

[4] Boudol, G., Castellani, I.: Permutation of transitions: An event structure semantics for CCS and SCCS. In: de Bakker, J.W., de Roever, W.P., Rozenberg, G. (eds.) Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency. LNCS, vol. 354, pp. 411–427. Springer (1989). https://doi.org/10.1007/BFb0013028

[5] Castellan, S., Hayman, J., Lasson, M., Winskel, G.: Strategies as concurrent processes. Electronic Notes in Theoretical Computer Science **308**, 87–107 (2014). https://doi.org/10.1016/j.entcs.2014.10.006

[6] Crafa, S., Varacca, D., Yoshida, N.: Compositional event structure semantics for the internal π-calculus. In: Caires, L., Vasconcelos, V.T. (eds.) CONCUR. LNCS, vol. 4703, pp. 317–332. Springer (2007). https://doi.org/10.1007/978-3-540-74407-8_22

[7] Crafa, S., Varacca, D., Yoshida, N.: Event Structure Semantics of Parallel Extrusion in the Pi-Calculus. In: Birkedal, L. (ed.) FoSSaCS. LNCS, vol. 7213, pp. 225–239. Springer (2012). https://doi.org/10.1007/978-3-642-28729-9_15

[8] Cristescu, I., Krivine, J., Varacca, D.: A compositional semantics for the reversible pi-calculus. In: LICS. pp. 388–397. IEEE Computer Society (2013). https://doi.org/10.1109/LICS.2013.45

[9] Cristescu, I., Krivine, J., Varacca, D.: Rigid families for the reversible π-calculus. In: Devitt, S.J., Lanese, I. (eds.) RC. LNCS, vol. 9720, pp. 3–19. Springer (2016). https://doi.org/10.1007/978-3-319-40578-0_1

[10] Danos, V., Krivine, J.: Reversible Communicating Systems. In: Gardner, P., Yoshida, N. (eds.) CONCUR. LNCS, vol. 3170, pp. 292–307. Springer (2004). https://doi.org/10.1007/978-3-540-28644-8_19

[11] Graversen, E., Phillips, I., Yoshida, N.: Towards a categorical representation of reversible event structures. Journal of Logical and Algebraic Methods in Programming **104**, 16 – 59 (2019). https://doi.org/10.1016/j.jlamp.2019.01.001

[12] Graversen, E., Phillips, I., Yoshida, N.: Event structures for the reversible early internal π-calculus. In: Lanese, I., Rawski, M. (eds.) RC. LNCS, vol. 12227, pp. 71 – 90. Springer (2020). https://doi.org/10.1007/978-3-030-52482-1_4

[13] Graversen, E., Phillips, I., Yoshida, N.: Event structure semantics of (controlled) reversible CCS. Journal of Logical and Algebraic Methods in Programming **121**, 100686 (2021). https://doi.org/10.1016/j.jlamp.2021.100686

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

[15] Honda, K., Yoshida, N.: On reduction-based process semantics. Theoretical Computer Science **151**(2), 437 – 486 (1995). https://doi.org/10.1016/0304-3975(95)00074-7

[16] Lanese, I., Medić, D., Mezzina, C.A.: Static versus dynamic reversibility in CCS. Acta Informatica **58**, 1–34 (2021). https://doi.org/10.1007/s00236-019-00346-6

[17] Lanese, I., Mezzina, C.A., Stefani, J.B.: Reversing Higher-Order Pi. In: Gastin, P., Laroussinie, F. (eds.) CONCUR. LNCS, vol. 6269, pp. 478–493. Springer (2010). https://doi.org/10.1007/978-3-642-15375-4_33

[18] Lanese, I., Mezzina, C.A., Stefani, J.B.: Reversibility in the higher-order π-calculus. Theoretical Computer Science **625**, 25 – 84 (2016). https://doi.org/10.1016/j.tcs.2016.02.019

[19] Lanese, I., Phillips, I., Ulidowski, I.: An axiomatic approach to reversible computation. In: Goubault-Larrecq, J., König, B. (eds.) FoSSaCS. LNCS, vol. 12077, pp. 442–461. Springer (2020). https://doi.org/10.1007/978-3-030-45231-5_23

[20] Langerak, R.: Transformations and Semantics for LOTOS. Ph.D. thesis, Universiteit Twente (1992), `https://books.google.dk/books?id=qB4EAgAACAAJ`

[21] Medić, D., Mezzina, C.A., Phillips, I., Yoshida, N.: A parametric framework for reversible π-calculi. Information and Computation **275**, 104644 (2020). https://doi.org/10.1016/j.ic.2020.104644

[22] Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, I and II. Information and Computation **100**(1), 1–77 (1992). https://doi.org/10.1016/0890-5401(92)90008-4

[23] Milner, R., Parrow, J., Walker, D.: Modal logics for mobile processes. Theoretical Computer Science **114**(1), 149–171 (1993). https://doi.org/10.1016/0304-3975(93)90156-N

[24] Milner, R., Sangiorgi, D.: Barbed bisimulation. In: Kuich, W. (ed.) ICALP. LNCS, vol. 623, pp. 685–695. Springer (1992). https://doi.org/10.5555/646246.684864

[25] Phillips, I., Ulidowski, I.: Reversing algebraic process calculi. Journal of Logic and Algebraic Programming **73**(1-2), 70–96 (2007). https://doi.org/10.1016/j.jlap.2006.11.002

[26] Phillips, I., Ulidowski, I.: Reversibility and models for concurrency. Electronic Notes in Theoretical Computer Science **192**(1), 93–108 (2007). https://doi.org/10.1016/j.entcs.2007.08.018

[27] Phillips, I., Ulidowski, I.: Reversibility and asymmetric conflict in event structures. Journal of Logical and Algebraic Methods in Programming **84**(6), 781 – 805 (2015). https://doi.org/10.1016/j.jlamp.2015.07.004

[28] Sangiorgi, D.: π-calculus, internal mobility, and agent-passing calculi. Theoretical Computer Science **167**(1), 235 – 274 (1996). https://doi.org/10.1016/0304-3975(96)00075-8

[29] Sewell, P., Wojciechowski, P.T., Unyapoth, A.: Nomadic pict: Programming languages, communication infrastructure overlays, and semantics for mobile computation. ACM Transactions on Programming Languages and Systems **32**(4), 12:1–12:63 (2010). https://doi.org/10.1145/1734206.1734209

[30] Tiezzi, F., Yoshida, N.: Reversible session-based pi-calculus. Journal of Logical and Algebraic Methods in Programming **84**(5), 684 – 707 (2015). https://doi.org/10.1016/j.jlamp.2015.03.004

[31] Winskel, G.: Event structure semantics for CCS and related languages. In: Nielsen, M., Schmidt, E.M. (eds.) ICALP. LNCS, vol. 140, pp. 561–576. Springer (1982). https://doi.org/10.1007/BFb0012800

[32] Winskel, G., Nielsen, M.: Models for concurrency. In: Handbook of Logic in Computer Science (Vol. 4), pp. 1–148. Oxford University Press, Inc. (1995)

A. Example of applying both event structures to a process

Example A.1. *Consider the process,* $a(x)|\overline{a}(b).\overline{b}(c)$*. We would like to show the correspondence between the LRPESs generated by this process using the two methods presented. Using the denotational event structure semantics of Section 6, we get the event structure* $\left\{\!\left[a(x)|\overline{a}(b).\overline{b}(c) \right]\!\right\}_{a,b,c,x} = \left\langle (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act}), \emptyset, \emptyset \right\rangle$ *shown in Figure A.5, where, since there is only one event per label, we assume the events have the same name and label for simplicity:*

$E = F = \{a(x), a(a), a(b), a(c), \tau, \overline{a}(b), \overline{b}(c)\}$

$\{\overline{a}(b)\} \mapsto a(b),\ \{\overline{a}(b), \tau\} \mapsto \overline{b}(c),\ \{\overline{b}(c)\} \mapsto a(c)$

$a(x) \sharp a(a),\ a(x) \sharp a(b),\ a(x) \sharp a(c),\ a(a) \sharp a(b),\ a(a) \sharp a(c),\ a(b) \sharp a(c),$
$a(x) \sharp \tau,\ a(a) \sharp \tau,\ a(b) \sharp \tau,\ a(c) \sharp \tau,\ \overline{a}(b) \sharp \tau$

$a(b) \rhd \underline{\overline{a}(b)},\ \overline{b}(c) \rhd \underline{\tau},\ \overline{b}(c) \rhd \underline{\overline{a}(b)},\ a(c) \rhd \underline{\overline{b}(c)}$

We then turn this LRBES into an LRPES, $P_B((E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})) = (E_P, F_P, <_P, \sharp_P, \prec_P, \rhd_P, \lambda_P, \mathsf{Act}_P)$*:*

$E_P = F_P = \{(\emptyset, a(x)), (\emptyset, a(a)), (\{\overline{a}(b)\}, a(b)), (\{\overline{b}(c), \overline{a}(b)\}, a(c)),$
$(\emptyset, \overline{a}(b)), (\emptyset, \tau), (\{\overline{a}(b)\}, \overline{b}(c)), (\{\tau\}, \overline{b}(c))\}$

$$\begin{aligned}
&\{(\emptyset, \overline{a}(b))\} <_P (\{\overline{a}(b)\}, a(b)), && (\emptyset, \overline{a}(b)) <_P (\{\overline{a}(b)\}, \overline{b}(c)),\\
&(\emptyset, \tau) <_P (\{\tau\}, \overline{b}(c)), && (\{\overline{a}(b)\}, \overline{b}(c)) <_P (\{\overline{b}(c), \overline{a}(b)\}, a(c)),\\
&(\emptyset, \overline{a}(b)) <_P (\{\overline{b}(c), \overline{a}(b)\}, a(c)) &&
\end{aligned}$$

$$\begin{aligned}
&(\emptyset, a(x)) \sharp_P (\emptyset, a(a)), && (\emptyset, a(x)) \sharp_P (\{\overline{a}(b)\}, a(b)),\\
&(\emptyset, a(x)) \sharp_P (\{\overline{b}(c), \overline{a}(b)\}, a(c)), && (\emptyset, a(a)) \sharp_P (\{\overline{a}(b)\}, a(b)),\\
&(\emptyset, a(a)) \sharp_P (\{\overline{b}(c), \overline{a}(b)\}, a(c)), && (\{\overline{a}(b)\}, a(b)) \sharp_P (\{\overline{b}(c), \overline{a}(b)\}, a(c)),\\
&(\emptyset, a(x)) \sharp_P (\emptyset, \tau), && (\emptyset, a(a)) \sharp_P (\emptyset, \tau),\\
&(\{\overline{a}(b)\}, a(b)) \sharp_P (\emptyset, \tau), && (\emptyset, \tau) \sharp_P (\{\overline{b}(c), \overline{a}(b)\}, a(c)),\\
&(\emptyset, \overline{a}(b)) \sharp_P (\emptyset, \tau), && (\emptyset, a(x)) \sharp_P (\{\tau\}, \overline{b}(c)),\\
&(\emptyset, a(a)) \sharp_P (\{\tau\}, \overline{b}(c)), && (\{\overline{a}(b)\}, a(b)) \sharp_P (\{\tau\}, \overline{b}(c)),\\
&(\{\overline{b}(c), \overline{a}(b)\}, a(c)) \sharp_P (\{\tau\}, \overline{b}(c)), && (\emptyset, \overline{a}(b)) \sharp_P (\{\tau\}, \overline{b}(c)),\\
&(\{\overline{a}(b)\}, \overline{b}(c)) \sharp_P (\emptyset, \tau), && (\{\overline{a}(b)\}, \overline{b}(c)) \sharp_P (\{\tau\}, \overline{b}(c))
\end{aligned}$$

$$\begin{aligned}
&(\{\overline{a}(b)\}, a(b)) \rhd_P \underline{(\emptyset, \overline{a}(b))}, && (\{\overline{a}(b)\}, \overline{b}(c)) \rhd_P \underline{(\emptyset, \overline{a}(b))},\\
&(\{\overline{b}(c), \overline{a}(b)\}, a(c)) \rhd_P \underline{(\emptyset, \overline{a}(b))}, && , (\{\tau\}, \overline{b}(c)) \rhd_P \underline{(\emptyset, \tau)}\\
&(\{\overline{b}(c), \overline{a}(b)\}, a(c)) \rhd_P \underline{(\{\overline{a}(b)\}, \overline{b}(c))} &&
\end{aligned}$$

$\lambda_P((X, e)) = \lambda(e)$

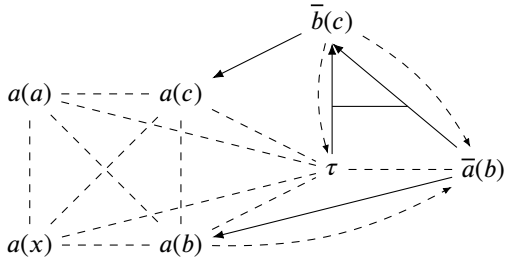

Figure A.5: The LRBES seen in Example A.1

Note that there are now two $\overline{b}(c)$ events, but only one $a(c)$ event, as $a(c)$ is in conflict with τ, and therefore cannot be caused by it, so we do not get $(\{\overline{b}(c), \tau\}, a(c))$.

We also generate another LRPES by applying the operational semantics of Section 7 to $(\emptyset, \emptyset, \emptyset) \vdash a(x)|\overline{a}(b).\overline{b}(c)$ to get a PES. Similarly to Example 7.5 we exclude traces requiring α-conversion and any inputs of names not already present in the process in order to get a finite number of events. We apply LR to turn it into a CLRPES, as seen in Definition 7.7, giving us $(E', F', <', \sharp', \prec', \rhd', \lambda', \mathsf{Act}')$ with $E' = F'$ consisting of:

$$
\begin{aligned}
e'_{a(x)} &= \left[\left(a(x), 0[a(x)][0], \emptyset\right)\right]_{\approx} \\
e'_{a(a)} &= \left[\left(a(a), 0[a(x)][0], \emptyset\right)\right]_{\approx} \\
e'_{a(b)} &= \left[\left(\overline{a}(b), 1[\overline{a}(b).\overline{b}(c)][\overline{b}(c)], \emptyset\right), \left(a(b), 0[a(x)][0], \{(\overline{a}(b), 1[\overline{a}(b).\overline{b}(c)][\overline{b}(c)])\}\right)\right]_{\approx} \\
e'_{a(c)} &= \left[\left(\overline{a}(b), 1[\overline{a}(b).\overline{b}(c)][\overline{b}(c)], \emptyset\right), \left(\overline{b}(c), 1[\overline{b}(c)][0], \{(\overline{a}(b), 1[\overline{a}(b).\overline{b}(c)][\overline{b}(c)])\}\right),\right. \\
&\qquad \left.\left(a(c), 0[a(x)][0], \{\overline{b}(c), 1[\overline{b}(c)][0]\}\right)\right]_{\approx} \\
e'_{\tau} &= \left[\left(\tau, \left\langle 0[a(x)][0], 1[\overline{a}(b)][0]\right\rangle, \emptyset\right)\right]_{\approx} \\
e'_{\overline{a}(b)} &= \left[\left(\overline{a}(b), 1[\overline{a}(b).\overline{b}(c)][\overline{b}(c)], \emptyset\right)\right]_{\approx} \\
e'_{\overline{b}(c)_{\overline{a}(b)}} &= \left[\left(\overline{a}(b), 1[\overline{a}(b).\overline{b}(c)][\overline{b}(c)], \emptyset\right),\right. \\
&\qquad \left.\left(\overline{b}(c), 1[\overline{b}(c)][0], \{(\overline{a}(b), 1[\overline{a}(b).\overline{b}(c)][\overline{b}(c)])\}\right)\right]_{\approx} \\
e'_{\overline{b}(c)_{\tau}} &= \left[\left(\tau, \left\langle 0[a(x)][0], 1[\overline{a}(b)][0]\right\rangle, \emptyset\right), \left(\overline{b}(c), 1[\overline{b}(c)][0], \emptyset\right)\right]_{\approx}
\end{aligned}
$$

where

$$
\begin{array}{lll}
e'_{\overline{a}(b)} <' e'_{a(b)}, & e'_{\overline{a}(b)} <' e'_{\overline{b(c)}_{\overline{a}(b)}}, & e'_{\tau} <' e'_{\overline{b(c)}_{\tau}}, \\
e'_{\overline{b(c)}_{\overline{a}(b)}} <' e'_{a(c)}, & e'_{\tau} <' e'_{a(c)} &
\end{array}
$$

$$
\begin{array}{lll}
e'_{a(x)} \sharp' e'_{a(a)}, & e'_{a(x)} \sharp' e'_{a(b)}, & e'_{a(x)} \sharp' e'_{a(c)}, \\
e'_{a(a)} \sharp' e'_{a(b)}, & e'_{a(a)} \sharp' e'_{a(c)}, & e'_{a(b)} \sharp' e'_{a(c)}, \\
e'_{a(x)} \sharp' e'_{\tau}, & e'_{a(a)} \sharp' e'_{\tau}, & e'_{a(b)} \sharp' e'_{\tau}, \\
e'_{a(c)} \sharp' e'_{\tau}, & e'_{a(c)} \sharp' e'_{\overline{b(c)}_{\overline{a}(b)}}, & e'_{\tau} \sharp e'_{\overline{a}(b)}, \\
e'_{a(x)} \sharp' e'_{\overline{b(c)}_{\tau}}, & e'_{a(a)} \sharp' e'_{\overline{b(c)}_{\tau}}, & e'_{a(b)} \sharp' e'_{\overline{b(c)}_{\tau}}, \\
e'_{a(c)} \sharp' e'_{\overline{b(c)}_{\tau}}, & e'_{\tau} \sharp' e'_{\overline{b(c)}_{\tau}}, & e'_{\overline{b(c)}_{\overline{a}(b)}} \sharp' e'_{\overline{b(c)}_{\tau}}
\end{array}
$$

$$
\begin{array}{lll}
e'_{a(b)} \rhd' \underline{e'_{\overline{a}(b)}}, & e'_{\overline{b(c)}_{\overline{a}(b)}} \rhd' \underline{e'_{\overline{a}(b)}}, & e'_{\overline{b(c)}_{\tau}} \rhd' \underline{e'_{\overline{b(c)}_{\tau}}}, \\
e'_{a(c)} \rhd' \underline{e'_{\overline{b(c)}_{\overline{a}(b)}}}, & e'_{\tau} \rhd' \underline{e'_{\overline{b(c)}_{\overline{a}(b)}}} &
\end{array}
$$

$$
\lambda(e') = \begin{cases} \mu & \text{if } e' = e'_{\mu} \text{ and } \mu \in \{a(x), a(a), a(b), a(c), \tau, \overline{a}(b)\} \\ \overline{b}(c) & \text{if } e' \in \{e'_{\overline{b(c)}_{\overline{a}(b)}}, e'_{\overline{b(c)}_{\tau}}\} \end{cases}
$$

$\mathsf{Act}' = \{a(x), a(a), a(b), a(c), \tau, \overline{a}(b), \overline{b}(c)\}$

We can define a mapping between these two event structures,

$$
f((X, e) = \begin{cases} e'_{e} & \text{if } e \in \{a(x), a(a), a(b), a(c), \tau, \overline{a}(b)\} \\ e'_{\overline{b(c)}_{\overline{a}(b)}} & \text{if } e = \overline{b}(c) \text{ and } \overline{a}(b) \in X \\ e'_{\overline{b(c)}_{\tau}} & \text{if } e = \overline{b}(c) \text{ and } \tau \in X \end{cases}
$$

and f and f^{-1} are both morphisms, since the event structures only differ in the names of the events.

However, as noted above, the real event structure generated by the operational semantics would also include events corresponding to traces in which α-conversion is performed, such as $[(\tau, \langle 0[a(y)][0], 1[\overline{a}(d)][0] \rangle, \emptyset)]_{\approx}$, which corresponds to the same action as e'_{τ}, but the communication is only performed after x has been α-converted to y and b to d. The denotational event structure semantics do not generate any event for this possibility, as they assume all names are distinct and the only α-conversion done is one necessitated by wanting to keep all names distinct after a replication.

B. Proofs from Section 2

B.1. Proof of Proposition 2.6

PROOF. 1. We prove this by induction on $\mathbf{H} \vdash P \xrightarrow[u]{\alpha} \mathbf{H}' \vdash Q$:

[SCOPE] In this case $P = (\nu x)P'$ and $Q = (\nu x)Q'$, $x \notin n(\alpha)$, and by induction $\mathbf{H}' \vdash Q' \overset{\alpha}{\underset{u}{\rightsquigarrow}} \mathbf{H} \vdash P'$. From rule [SCOPE^{-1}] we therefore get $\mathbf{H}' \vdash Q \overset{\alpha}{\underset{u}{\rightsquigarrow}} \mathbf{H} \vdash P$.

[PAR$_i$] In this case $P = P_0|P_1$ and $Q = Q_0|Q_1$, $P_{1-i} = Q_{1-i}$, in $\alpha = \overline{a}(n)$ then $n \notin \mathsf{fn}(P_{1-i})$, and by induction we have a transition $([\check{i}]\overline{H}', [\check{i}]\underline{H}', [\check{i}]H') \vdash Q_i \overset{\alpha}{\underset{u}{\rightsquigarrow}} ([\check{i}]\overline{H}, [\check{i}]\underline{H}, [\check{i}]H) \vdash P_i$, meaning according to rule [PAR$_i^{-1}$], $\mathbf{H}' \vdash Q \overset{\alpha}{\underset{u}{\rightsquigarrow}} \mathbf{H} \vdash P$.

[COM$_i$] In this case $P = P_0|P_1$ and $Q = Q_0|Q_1$, $n \notin \mathsf{fn}(P_j)$, $\overline{H} = \overline{H}'$, $\underline{H} = \underline{H}'$, and $H' = H \cup \{((\alpha_0, \alpha_1), (0v_0, 1v_1))\}$. By induction and Lemma 2.5, we get transitions $([\check{i}]\overline{H}', [\check{i}]\underline{H}', [\check{i}]H') \vdash Q_i \overset{\alpha_i}{\underset{v_i}{\rightsquigarrow}} ([\check{i}]\overline{H}, [\check{i}]\underline{H}, [\check{i}]H) \vdash P_i$ and $([\check{j}]\overline{H}', [\check{j}]\underline{H}', [\check{j}]H') \vdash Q_j \overset{\alpha_j}{\underset{v_j}{\rightsquigarrow}} ([\check{j}]\overline{H}, [\check{j}]\underline{H}, [\check{j}]H) \vdash P_j$. This means according to [COM^{-1}], $\mathbf{H}' \vdash Q \overset{\tau}{\underset{u}{\rightsquigarrow}} \mathbf{H} \vdash P$.

[STR] In this case $Q \equiv Q'$,$\mathbf{H}' \vdash Q' \overset{\alpha}{\underset{u}{\rightsquigarrow}} \mathbf{H} \vdash P'$, and $P' \equiv P$, and by rule [STR^{-1}], $\mathbf{H}' \vdash Q \overset{\alpha}{\underset{u}{\rightsquigarrow}} \mathbf{H} \vdash P$.

[OUT] In this Case $P = \sum_{i \in I} \alpha_i.P_i$, $Q = P_j$ and $\alpha = \overline{a}(n) = \alpha_j$ for some $j \in I$, and by [OUT^{-1}], $\mathbf{H}' \vdash Q \overset{\alpha}{\underset{u}{\rightsquigarrow}} \mathbf{H} \vdash P$.

[IN] Similar to [OUT].

2. Similar to previous.

B.2. Proof of Proposition 2.10

PROOF. This proof is similar to Theorem 14 of [14]. Either we have a path l such that $u_i = l0u_i'$ and $u_{1-i} = l1u_{1-i}'$, or we have $\mu_i = \tau$.

In the first case, if $Q|R$ is the parallel composition at location l, then we have transitions $([l\check{0}]\overline{H}, [l\check{0}]\underline{H}, [l\check{0}]H) \vdash Q \overset{\alpha_i}{\underset{u_i'}{\rightarrowtail}}$ and $([l\check{1}]\overline{H}, [l\check{1}]\underline{H}, [l\check{1}]H) \vdash Q \overset{\alpha_{1-i}}{\underset{u_{1-i}'}{\rightarrowtail}}$ and there does not exist n such that $D_i(n) = u_{1-i}$ or there does not exist n such that $D_{1-i}(n) = u_i$, and by [PAR$_i$] and [PAR$_i^-$1], this means $\mathbf{H}_0 \vdash P_0 \xrightarrow[u_1]{\alpha_1} \mathbf{H}_0' \vdash P'$ and $\mathbf{H}_1 \vdash P_1 \xrightarrow[u_0]{\alpha_0} \mathbf{H}_1' \vdash P'$ and by Lemma 15 of [14], $\mathbf{H}_0' = \mathbf{H}_1'$. This goes for reverse transitions too.

If $\mu_i = \tau$ then $u_i = l_i \langle l_{i0}[P_0][Q_0], l_{i1}[P_1][Q_1] \rangle$ and we can split the transition into $l_{i0}[P_0][Q_0]$ and $l_{i1}[P_1][Q_1]$ at the parallel composition on location l and use similar logic.

B.3. Proof of Lemma 2.12

PROOF. If $\overset{\alpha_0}{\underset{u_0,D_0}{\rightsquigarrow}}$ and $\overset{\alpha_1}{\underset{u_1,D_1}{\rightsquigarrow}}$ are not independent, then either u_0 and u_1 are not independent, or there exists n such that $D_i(n) = u_{1-i}$.

Obviously, if u_0 and u_1 are not independent, then $u_0 = u_1$ and $\alpha_0 = \alpha_1$, $D_0 = D_1$, and $\mathbf{H}_0 \vdash P_0 \equiv \mathbf{H}_1 \vdash P_1$ follows from that.

If u_0 and u_1 are independent and exists n such that $D_i(n) = u_{1-i}$, then by $[\mathrm{PAR}_i^{-1}]$, we cannot have $\mathbf{H} \vdash P \overset{\alpha_{1-i}}{\underset{u_{1-i},D_{1-i}}{\rightsquigarrow}} \mathbf{H}_{1-i} \vdash P_{1-i}$.

C. Proofs from Section 3

C.1. Proof of Lemma 3.4

PROOF. This would require $\overline{b}(x)[m]$ to either prefix, be in parallel with, or be an alternative choice to $\overline{a}(x)$ in P. The first two cases are impossible due to the "if $\mu = \overline{a}(x)$ then $x \notin n(\alpha)$" and "if $\mu = \overline{a}(x)$ then $x \notin \mathsf{fn}(P_1)$" requirements in the rules for propagating $\overline{a}(x)[n]$ past past actions and parallel composition, and the last case is prevented by requiring alternative paths to be standard if we want to propagate an action past the choice.

C.2. Proof of Proposition 3.5

PROOF. 1. We perform induction on $P \xrightarrow{\mu[n]} Q$:

(a) Suppose $P = a(x).P'$, $\mu = a(b)$, $\mathsf{std}(P')$, $Q = a(b)[n].Q'$, and $Q' = Q[x := b_{[n]}]$. Then, since $x \notin n(Q')$, $Q \overset{a(b)}{\rightsquigarrow} P$.

(b) Suppose $P = \overline{a}(x).P'$, $\mu = \overline{a}(x)$, $\mathsf{std}(P')$, $Q = \overline{a}(x)[n].P'$. Then clearly $Q \overset{\overline{a}(x)}{\rightsquigarrow} P$.

(c) Suppose $P = \alpha[m].P'$, $P' \xrightarrow{\mu[n]} Q'$, $Q = \alpha[m].Q'$, $n \neq m$, and if $\mu = \overline{a}(x)$ then $x \notin n(\alpha)$. Then by induction $Q' \overset{\mu[n]}{\underset{P}{\rightsquigarrow}} '$, and clearly $Q \overset{\mu[n]}{\rightsquigarrow} P$.

(d) Suppose $P = P_0|P_1$, $P_0 \xrightarrow{\mu[n]} Q_0$, $\mathsf{fsh}[n](P_1)$, $Q = Q_0|P_1$, and if $\mu = \overline{a}(x)$ then $x \notin \mathsf{fn}(P_1)$. Then by induction, $Q_0 \overset{\mu[n]}{\rightsquigarrow} P_0$, and obviously $Q \overset{\mu[n]}{\rightsquigarrow} P$.

(e) Suppose $P = P_0|P_1$, $P_0 \xrightarrow{a(x)[n]} Q_0$, $P_1 \xrightarrow{\overline{a}(x)[n]} Q_1$, $\mu = \tau$, and $Q = (\nu x)(Q_0|Q_1)$. Then by induction $Q_0 \overset{a(x)}{\rightsquigarrow} P_0$ and $Q_1 \overset{\overline{a}(x)}{\rightsquigarrow} P_1$, meaning clearly $Q \overset{\mu[n]}{\rightsquigarrow} P$.

(f) Suppose $P = P_0 + P_1$, $P_0 \xrightarrow{\mu[n]} Q_0$, $\mathsf{std}(P_1)$, and $Q = Q_0 + P_1$. Then by induction $Q_0 \overset{\mu[n]}{\rightsquigarrow} P_0$, meaning $Q \overset{\mu[n]}{\rightsquigarrow} P$.

(g) Suppose $P = (\nu x)P'$, $P' \xrightarrow{\mu[n]} Q'$, $x \notin n(\mu)$, and $Q = (\nu x)Q'$. Then by induction $Q' \overset{\mu[n]}{\rightsquigarrow} P'$, and we get $Q \overset{\mu[n]}{\rightsquigarrow} P$.

(h) Suppose $P \equiv P'$, $P' \xrightarrow{\mu[n]} Q'$, and $Q \equiv Q'$. Then by induction $Q' \overset{\mu[n]}{\rightsquigarrow} P'$, and therefore $Q \overset{\mu[n]}{\rightsquigarrow} P$.

2. We prove this by induction on $P \overset{\mu[n]}{\rightsquigarrow} Q$:
   (a) Suppose $P = a(b)[n].P'$, $\mu = a(b)$, $\mathsf{std}(P')$, $x \notin n(P')$, $Q' = P'[b_{[n]} := x]$, and $Q = a(x).Q'$. Then clearly $Q \xrightarrow{\mu[n]} P$.
   (b) Suppose $P = \overline{a}(x)[n].P'$, $\mu = \overline{a}(x)$, $\mathsf{std}(P')$, $Q = \overline{a}(x).P'$. Then clearly $Q \xrightarrow{\overline{a}(x)} P$.
   (c) Suppose $P = \alpha[m].P'$, $P' \overset{\mu[n]}{\rightsquigarrow} Q'$, $m \neq n$, and $Q = \alpha[n].Q'$. Then by induction, $Q' \xrightarrow{\mu[n]} P'$, and since P is forwards reachable, if $\mu = \overline{a}(x)$ then $x \notin n(\alpha)$. This means $Q \xrightarrow{\mu[n]} P$.
   (d) Suppose $P = P_0|P_1$, $P_0 \overset{\mu[n]}{\rightsquigarrow} Q_0$, $\mathsf{fsh}[n](P_1)$, $Q = Q_0|P_1$, and if $\mu = \overline{a}(x)$ then $x \notin \mathsf{fn}(P_1)$. Then by induction $Q_0 \xrightarrow{\mu[n]} P_0$, and clearly $Q \xrightarrow{\mu[n]} P$.
   (e) Suppose $P = (\nu x)(P_0|P_1)$, $\mu = \tau$, $P_0 \overset{a(x)[n]}{\rightsquigarrow} Q_0$, $P_1 \overset{\overline{a}(x)[n]}{\rightsquigarrow} Q_1$, and $Q = Q_0|Q_1$. Then by induction $Q_0 \xrightarrow{a(x)} P_0$ and $Q_1 \xrightarrow{\overline{a}(x)} P_1$, meaning clearly $Q \xrightarrow{\mu[n]} P$.
   (f) Suppose $P = P_0 + P_1$, $P_0 \overset{\mu[n]}{\rightsquigarrow} Q_0$, $\mathsf{std}(P_1)$, and $Q = Q_0 + P_1$. Then by induction $Q_0 \xrightarrow{\mu[n]} P_0$, meaning $Q \xrightarrow{\mu[n]} P$.
   (g) Suppose $P = (\nu x)P'$, $P' \overset{\mu[n]}{\rightsquigarrow} Q'$, $x \notin n(\mu)$, and $Q = (\nu x)Q'$. Then by induction $Q' \xrightarrow{\mu[n]} P'$, and we get $Q \xrightarrow{\mu[n]} P$.
   (h) Suppose $P \equiv P'$, $P' \overset{\mu[n]}{\rightsquigarrow} Q'$, and $Q \equiv Q'$. Then by induction $Q' \xrightarrow{\mu[n]} P'$, and therefore $Q \xrightarrow{\mu[n]} P$.

C.3. Proof of Proposition 3.6

PROOF. We use structural induction on P to prove both these at once:

1. Suppose $P = 0$ or $P = \alpha.P'$. Then P cannot do any backwards transitions.
2. Suppose $P = \alpha[n'].P'$. Then either $\mathsf{std}(P')$ and $n = m = n'$, or $Q = a(b)[n'].Q'$, $R = a(b)[n'].R'$, $P' \overset{\mu[m]}{\rightsquigarrow} Q'$, and $P' \overset{\mu'[n]}{\rightsquigarrow} R'$, meaning by induction there exists S' such that $Q' \overset{\mu'[n]}{\rightsquigarrow} S'$ and $R' \overset{\mu[m]}{\rightsquigarrow} S'$. We say that $S = \alpha[n].S'$, and the property holds.
3. Suppose $P = P_0 + P_1$, then either $\mathsf{std}(P_0)$, $P_1 \overset{\mu[m]}{\rightsquigarrow} Q_1$, $P \overset{\mu'[n]}{\rightsquigarrow} R_1$, $Q = P_0 + Q_1$, and $R = P_0 + R_1$, or $\mathsf{std}(P_1)$, $P_0 \overset{\mu[m]}{\rightsquigarrow} Q_0$, $P \overset{\mu'[n]}{\rightsquigarrow} R_0$, $Q = Q_0 + P_1$, and $R = R_0 + P_1$. In the first case, by induction there exists an S_1 such that $Q_1 \overset{\mu'[n]}{\rightsquigarrow} S_1$ and $R_1 \overset{\mu[m]}{\rightsquigarrow} S_1$, and we define $S = P_0 + S_1$, and the property holds. The second case is similar.

4. Suppose $P = (\nu x)P'$. Then either (1) $P' \xrightarrow{\mu[m]} Q'$ and $x \notin n(\mu)$ and $Q = (\nu x)Q'$ or (2) $P' = P_0|P_1$, $P_i \xrightarrow{a(x)[m]} Q_i$, $P_{1-i} \xrightarrow{\overline{a}(x)[m]} Q_{i-1}$, $\mu = \tau$, and $Q = Q_0|Q_1$, and either (a) $P' \xrightarrow{\mu'[n]} R'$ and $x \notin n(\mu')$ and $R = (\nu x)R'$ or (b) $P' = P_0|P_1$, $P_i \xrightarrow{a(x)[n]} R_i$, $P_{1-i} \xrightarrow{\overline{a}(x)[n]} R_{i-1}$, $\mu' = \tau$, and $R = R_0|R_1$.
   In case 1a, by induction there exists S' such that $Q' \xrightarrow{\mu'[n]} S'$ and $R' \xrightarrow{\mu[m]} S'$, and we define $S = (\nu x)S'$, and the property holds.
   In case 1b, there exists P_j such that $P_j \xrightarrow{\mu[m]} Q_j$, and $\mathsf{fsh}[m](P_{1-j})$, and if $\mu = \overline{a}(x)$ then $x \notin \mathsf{fn}(P_1)$. If $j = i$ then by induction there exists an S_i such that $Q_j \xrightarrow{\mu'[n]} S_i$ and $R_i \xrightarrow{a(x)[m]} S_i$, and we define $S = S_i|R_{1-i}$, and the property holds. If $I = 1 - j$, the argument is similar.
   Case 2a is similar to case 1b.
   Case 2b cannot occur because we cannot have more than one past action outputting the same name according to Lemma 3.4.
5. Suppose $P = P_0|P_1$. Then there exists an i such that either $P_i \xrightarrow{\mu[m]} Q_i$ and $P_i \xrightarrow{\mu'[n]} R_i$ and $Q = Q_i|P_{1-i}$ and $R = R_i|P_{1-i}$, or $P_i \xrightarrow{\mu[m]} Q_i$ and $P_{1-i} \xrightarrow{\mu'[n]} R_{1-i}$ and $Q = Q_i|P_{1-i}$ and $R = P_i|R_{1-i}$.
   In the first case, there exists S_i such that $Q_i \xrightarrow{\mu'[n]} S_i$ and $R_i \xrightarrow{\mu[m]} S_i$, and we define $S = S_i|P_{1-i}$ and the property holds.
   If the second case we define $S = Q_i|R_{1-i}$, and the property holds.

C.4. Proof of Proposition 3.7

PROOF. We prove this by structural induction:

1. Suppose $P = 0$ or $P = \alpha.P'$. Then P cannot do any reverse transitions.
2. Suppose $P = \alpha[n].P'$. Then either $\mathsf{std}(P')$, meaning $\mu = \mu' = \alpha$, $n = m$, and $Q \equiv R$, or $P' \xrightarrow{\mu[m]} Q'$, $P' \xrightarrow{\mu'[m]} R'$, $Q = \alpha[n].Q'$, and $R = \alpha[n].R'$, and the result follows from induction.
3. Suppose $P = P_0 + P_1$. Then the result follows from induction.
4. Suppose $P = (\nu x)P'$. Then either (1) $P' \xrightarrow{\mu[m]} Q'$ and $x \notin n(\mu)$ and $Q = (\nu x)Q'$ or (2) $P' = P_0|P_1$, $P_i \xrightarrow{a(x)[m]} Q_i$, $P_{1-i} \xrightarrow{\overline{a}(x)[m]} Q_{i-1}$, $\mu = \tau$, and $Q = Q_0|Q_1$, and either (a) $P' \xrightarrow{\mu'[m]} R'$ and $x \notin n(\mu')$ and $R = (\nu x)R'$ or (b) $P' = P_0|P_1$, $P_i \xrightarrow{a(x)[m]} R_i$, $P_{1-i} \xrightarrow{\overline{a}(x)[m]} R_{i-1}$, $\mu' = \tau$, and $R = R_0|R_1$.
   In case 1a the result follows from induction.
   In case 1b P_j such that $P_j \xrightarrow{\mu[m]} Q_j$, and $\mathsf{fsh}[m](P_{1-j})$, contradicting $P_{1-j} \xrightarrow{\alpha[m]} R_{1-j}$. Meaning this case cannot occur.
   Similar for case 2a.
   Case 2b follows from induction.

5. Suppose $P = P_0|P_1$. Then there exists an i such that either $P_i \overset{\mu[m]}{\rightsquigarrow} Q_i$ and $P_i \overset{\mu'[m]}{\rightsquigarrow} R_i$ and $Q = Q_i|P_{1-i}$ and $R = R_i|P_{1-i}$, or $P_i \overset{\mu[m]}{\rightsquigarrow} Q_i$ and $P_{1-i} \overset{\mu'[m]}{\rightsquigarrow} R_{1-i}$ and $Q = Q_i|P_{1-i}$ and $R = P_i|R_{1-i}$.

   In the first case the result follows from induction. In the second case $P_i \overset{\mu[m]}{\rightsquigarrow} Q_i$ requires $\mathsf{fsh}[m](P_{1-i})$, which contradicts $P_{1-i} \overset{\mu'[m]}{\rightsquigarrow} R_{1-i}$, meaning this case cannot occur.

C.5. Proof of Theorem 3.9

PROOF. We say that $P \overset{\mu_0[m_0]}{\rightarrowtail} P_0 \ldots \overset{\mu_n[m_n]}{\rightarrowtail} P_n = Q$ and perform induction on the length of the trace, the number of pairs $\xrightarrow{\mu_i[m_i]} \overset{\mu_{i+1}[m_{i+1}]}{\rightsquigarrow}$ in the trace, and the location of the first such pair.

If no such pair exists then R must exist.

Otherwise, we say that $\xrightarrow{\mu_i[m_i]} \overset{\mu_{i+1}[m_{i+1}]}{\rightsquigarrow}$ is the first such pair in the trace. We have two cases, either $m_i = m_{i+1}$ or not.

If $m_i = m_{i+1}$ then by Propositions 3.5 and 3.7, $P_{i-1} = P_{i+1}$, and we therefore have a trace $P \overset{\mu_0[m_0]}{\rightarrowtail} P_0 \ldots \overset{\mu_{i-1}[m_{i-1}]}{\rightarrowtail} P_{i-1} \overset{\mu_{i+2}[m_{i+2}]}{\rightarrowtail} \ldots \overset{\mu_n[m_n]}{\rightarrowtail} P_n = Q$.

If $m_i \neq m_{i+1}$ then by Lemma 3.8, P_i is forwards-reachable, and therefore by Proposition 3.5, we get a transition $P_i \overset{\mu_i[m_i]}{\rightsquigarrow} P_{i-1}$, and by Proposition 3.6 we have a trace $P \overset{\mu_0[m_0]}{\rightarrowtail} P_0 \ldots P_{i-1} \overset{\mu_{i+1}P_i'[m_{i+1}]}{\rightsquigarrow} \xrightarrow{\mu_i[m_i]} P_{i+1} \ldots \overset{\mu_n[m_n]}{\rightarrowtail} P_n = Q$

C.6. Proof of Proposition 3.13

PROOF. We prove both simultaneously by structural induction on P.

- Suppose $P = 0$. Then P cannot perform any actions, and the results are trivial.
- Suppose $P = a(x).P'$. Then $\mu = a(b)$ for some b, $Q = a(b)[n].P'[x := b]$, $\mu' = a(c)$ for some c, $R = a(c)[m].P'[x := c]$, and since no subsequent action can change b and c,we get $b = c$, and the results hold.
- Suppose $P = \overline{a}(x).P'$. Then $\mu = \overline{a}(x)$ $Q = \overline{a}(x)[n].P'$, $\mu' = \overline{a}(x)$, $R = \overline{a}(x)[m].P'\overline{a}(x)$, and the results hold.
- Suppose $P = \alpha[n].P'$. Then the results follow from induction.
- Suppose $P = P_0 + P_1$. Then the results follow from induction, and the fact that both actions must use the same branch of the choice.
- Suppose $P = (\nu x)P'$. Then the result follows from induction.

- Suppose $P = P_0|P_1$. Then there exists an i such that either (1) $P_i \xrightarrow{\mu[m]} Q_i$ and $Q = Q_i|P_{1-i}$ or (2) $P_i \xrightarrow{a(x)[m]} Q_i$, $P_{1-i} \xrightarrow{\overline{a}(x)[m]} Q_{i-1}$, $\mu = \tau$, and $Q = Q_0|Q_1$ and there exists a j such that either (a) $P_j \xrightarrow{\mu'[n]} R_j$ and $R = R_j|P_{1-j}$ or (b) $P_j \xrightarrow{b(y)[n]} R_j$, $P_{1-j} \xrightarrow{\overline{b}(y)[n]} R_{j-1}$, $\mu' = \tau$, and $R = R_0|R_1$

  In Case 1a, either $i = j$ or $i \neq j$. If $i = j$ then the result follows from induction. If $i \neq j$, since we require $\mathsf{fsh}[m](P_{1-i})$ and $\mathsf{fsh}[n](P_{j-i})$, some where in $R \to^* T$ and $Q \to^* T$ respectively, $m \neq n$. For similar reasons, we cannot have that $\mu = \overline{a}(x)$ and $x \in n(\mu')$ or vice versa, meaning $Q \xrightarrow{\mu'[n]} S$ and $R \xrightarrow{\mu[m]} S$ and $S \to^*$.

  In Case 1b, suppose $i = j$. Then by induction, there exists S_i such that $Q_i \xrightarrow{b(y)[n]} S_i$, requiring $\mathsf{fsh}[n](R_{1-i})$ meaning $n \neq m$, and $R_i \xrightarrow{\mu[m]} S_i$, meaning we can define $S = S_i|R_{1-i}$, and requiring $\mathsf{fsh}[n](R_{1-i})$, and the result holds for that. If $i \neq j$ then the result is similar.

  Case 2a is similar.

  Case 2b follows from induction.

- Suppose $P = !P'$. Then $!P'|P' \xrightarrow{\mu[m]} Q$, and $!P'|P' \xrightarrow{\mu'[n]} R$, and this falls under the previous case.

C.7. Proof of Proposition 4.9

PROOF. By Theorem 4.8, $E(\mathsf{lcopy}(\mathbf{H}_P) \vdash P, P) \xrightarrow{\mu_0[m_0]} E(\mathsf{lcopy}(\mathbf{H}_Q) \vdash Q, Q)$ and $E(\mathsf{lcopy}(\mathbf{H}_P) \vdash P, P) \xrightarrow{\mu_1[m_1]} E(\mathsf{lcopy}(\mathbf{H}_R) \vdash R, R)$ and $E(\mathsf{lcopy}(\mathbf{H}_Q) \vdash Q, Q) \to^* E(\mathsf{lcopy}(\mathbf{H}_T) \vdash T, T)$ and $E(\mathsf{lcopy}(\mathbf{H}_R) \vdash R, R) \to^* E(\mathsf{lcopy}(\mathbf{H}_T) \vdash T, T)$. By Proposition 3.13, we get an S' such that $E(\mathsf{lcopy}(\mathbf{H}_Q) \vdash Q, Q) \xrightarrow{\mu_1[m_1]} S'$ and $E(\mathsf{lcopy}(\mathbf{H}_R) \vdash R, R) \xrightarrow{\mu_0[m_0]} S'$ and by Theorem 4.8 we get $\mathbf{H}_Q \vdash Q \xrightarrow[u_1']{\mu_1} \mathbf{H}_S \vdash S$ and $\mathbf{H}_R \vdash R \xrightarrow[u_0']{\mu_0} \mathbf{H}'_S \vdash S'$ such that $E(\mathsf{lcopy}(\mathbf{H}_S) \vdash S, S) \equiv E(\mathsf{lcopy}(\mathbf{H}'_S) \vdash S', S')$, and by Lemma 2.12, $u_0 = u_0'$ and $u_1 = u_1'$. And by Theorem 4.8 and Proposition 3.13, $\mathbf{H}_S \vdash S \to^* \mathbf{H}_T \vdash T$ and $\mathbf{H}'_S \vdash S' \to^* \mathbf{H}_T \vdash T$.

D. Proofs from Section 4

D.1. Proof of Lemma 4.4

PROOF. We prove this by structural induction on P:

- Assume $P = 0$. Then $P' = P[x_1 := a_1][x_2 := a_2] \dots [x_k := a_k] = 0$ and $S(P[x_1 := a_1][x_2 := a_2] \dots [x_k := a_k], P, [n], x_1) = 0$.

- Assume $P = b(c).Q$ Then either $P' = d(e).Q'$, or $P' = d(e)[m].Q'$, for some d, e, m. We then get 4 cases: either $b = x_1$, $c = x_1$, $b = c = x_1$, or $b \neq x_1$ and $c \neq x_1$.

  Assume $b = x_1$ and $c \neq x_1$. Then $S(P', P, [n], x_1) = d[n](c).S(Q', Q, [n], x_1)$ and $d = a_1$, and the result follows from induction.

  Assume $c = x_1$ and $b \neq x_1$. Then, since c is bound, $P[x_1 := a_1] = P[x_1 := a_{1_{\lfloor n \rfloor}}] = P$, and the result follows.

  Assume $b = c = x_1$. Then $d = a_1$ and $Q[x_1 := a_1][x_2 := a_2] \dots [x_k := a_k] = Q[x_2 := a_2] \dots [x_k := a_k] = Q'$, and the result follows.

- Assume $P = \overline{b}(c).Q$. This is similar to the previous case.

- Assume $P = \sum_{i \in I} P_i$. Then the result follows trivially from induction.

- Assume $P = P_0 | P_1$. Then either $P' = P'_0 | P'_1$, or $P_0 \equiv !P_1$ and $P' = P'_0$.

  If $P' = P'_0 | P'_1$ then the result follows trivially from induction.

  If $P_0 = !P_1$ and $P' = P_0$, then $P'' = S\left(!P'_0, P_0, [n], x\right) | S\left(P'_0, P_1, [n], x\right)$, and the result follows from induction.

- Assume $P = (\nu b)Q$. Then $P' = (\nu c)Q'$ and either $b = x_1$ or $b \neq x_1$.

  If $b = x_1$, then $P[x_1 := a_1] = P[x_1 := a_{1_{\lfloor n \rfloor}}] = P$.

  If $b \neq x_1$, then the result follows from induction.

- Assume $P = !Q$. Then either $P' = !Q'$, or $P' = P'_0 | P'_1$.

  If $P' = !Q'$, the result follows trivially from induction.

  Otherwise the case is similar to the second case on parallel composition.

D.2. Proof of Theorem 4.8

PROOF. We first show that if there exists a location u such that $\mathbf{H} \vdash P \xrightarrow[u]{\mu} \mathbf{H}' \vdash P'$, then $E(\mathsf{lcopy}(\mathbf{H}) \vdash P, P) \overset{\mu[m]}{\underset{E}{\rightarrowtail}} (\mathsf{lcopy}(\mathbf{H}') \vdash P', P')$ by induction on the size of $\overline{H} \cup \underline{H} \cup H$ and the structure of P:

Assume $\mathbf{H} = (\emptyset, \emptyset, \emptyset)$. Then $E(\mathbf{H} \vdash P, P) = P$.

Assume$\mathbf{H} \neq (\emptyset, \emptyset, \emptyset)$. Then we perform structural induction on P.

- Assume $P = a(x).Q$. Then $\mu = a(b)$, $u = [P][Q[x := b]]$, and $\mathbf{H}' \vdash P' = (\emptyset, \{(a(b), u)\}, \emptyset) \vdash Q[x := b]$. We then by Lemma 4.4 get $E(\mathbf{H}' \vdash P', P') = a(x)\left[[P][Q[x := b]]\right].Q[x := b_{[[P][Q[x:=b]]]}]$, and the rest of the case follows naturally.

- Assume $P = \overline{a}(x).Q$. This case is similar to the previous.

- Assume $P = P_0|P_1$. Then either $u = iu'$, or $u = \langle 0u_0, 1u_1\rangle$.

  If $u = 0u'$, then $(\emptyset,\emptyset,\emptyset) \vdash P_0 \xrightarrow[u']{\mu} \mathbf{H}'_0 \vdash P'_0$, $\mathbf{H}' \vdash P' = (0\overline{H'}_0, 0\underline{H'}_0, 0H'_0) \vdash P'_0|P_1$, and if $\mu = \overline{a}(b)$ then $b \notin \mathsf{fn}(P_1)$. By induction we have a transition $P_0 \xrightarrow{\mu[u']} E(\mathbf{H}'_0 \vdash P'_0, P_0)$, and therefore $P_0|P_1 \xrightarrow{\mu[u]} E(\mathbf{H}'_0 \vdash P'_0, P_0)|P_1 = E((0\overline{H'}_0, 0\underline{H'}_0, 0H'_0) \vdash P'_0|P_1, P'_0|P_1)$.
  If $u = 1u'$, the case is similar to $u = 0u'$.

  If $u = \langle 0u_0, 1u_1\rangle$, then $(\emptyset,\emptyset,\emptyset) \vdash P_i \xrightarrow[u_i]{a(b)} \mathbf{H}'_i \vdash P'_i$ and $(\emptyset,\emptyset,\emptyset) \vdash P_{1-i} \xrightarrow[u_{1-i}]{\overline{a}(b)} \mathbf{H}_{1-i} \vdash P'_{1-i}$ for some $i \in \{0,1\}$ and $\mathbf{H}' \vdash P' = (\emptyset,\emptyset,\{((a(b),\overline{a}(b)),u)\}) \vdash P'_0|P'_1$ and $b \notin \mathsf{fn}(P_i)$. By induction, $E((\emptyset,\emptyset,\emptyset) \vdash P_i, P_i) \xrightarrow{a(b)[u_i]} E(\mathbf{H}'_i \vdash P'_i, P'_i)$ and $E((\emptyset,\emptyset,\emptyset) \vdash P_{1-i}, P_{1-i}) \xrightarrow{\overline{a}(b)[u_{1-i}]} E(\mathbf{H}'_{1-i} \vdash P'_{1-i}, P'_{i-1})$. Therefore we have a transition $P_0|P_1 \xrightarrow{\tau[u]} E((\emptyset,\emptyset,\{((a(b),\overline{a}(b)),\langle 0u_0, 1u_1\rangle, m)\}) \vdash (\nu b)(P'_0|P'_1), (\nu b)(P'_0|P'_1)) =$
  $(\nu b)E((\emptyset,\emptyset,\{(a(b),\overline{a}(b),\langle 0u_0, 1u_1\rangle, m)\}) \vdash (P'_0|P'_1), (P'_0|P'_1))$.

- Assume $P = (\nu x)Q$. Then $(\emptyset,\emptyset,\emptyset) \vdash Q \xrightarrow[u]{\mu} \mathbf{H}' \vdash Q'$, $x \notin n(\mu)$, and $P' = (\nu x)Q'$. We then get by induction $Q \xrightarrow{\mu[u]} E(\mathbf{H}' \vdash Q', Q')$, and therefore $(\nu x)Q \xrightarrow{\mu[u]} (\nu x)E(\mathbf{H}' \vdash Q', Q') = E(\mathbf{H}' \vdash (\nu x)Q', (\nu x)Q')$.

- Assume $P = !Q$. Then $(\emptyset,\emptyset,\emptyset) \vdash !Q|Q \xrightarrow[u]{\mu} \mathbf{H}' \vdash P'$, and the rest follows from the parallel case.

If for any $(\mu', u') \in \overline{H} \cup \underline{H} \cup H$, if there exists a location u such that $\mathbf{H} - (\mu', u') \vdash P \xrightarrow[u]{\mu} \mathbf{H}'' \vdash P'$, then there exists a key m, such that $E(\mathbf{H} - (\mu', u') \vdash P, P) \xrightarrow{\mu[u]} E(\mathbf{H}'' \vdash P', P')$, then E only adds past actions and unused choice branches to the process, both of which one can easily propagate the action past.

We then show that if $E(\mathsf{lcopy}(\mathbf{H}) \vdash P, P) \xrightarrow{\mu[n]} Q$ then there exists a location, u and a πIH process, $\mathbf{H}' \vdash P'$, such that $\mathbf{H} \vdash P \xrightarrow[u]{\mu} \mathbf{H}' \vdash P'$ and $P'' \equiv E(\mathsf{lcopy}(\mathbf{H}') \vdash P', P')$. We again do this by induction on the number of extrusions in $\overline{H} \cup \underline{H} \cup H$, and the structure of P.

Assume $\overline{H} \cup \underline{H} \cup H = \emptyset$. Then $E(\mathsf{lcopy}(\mathbf{H}) \vdash P, P) = P$. Since we are only proving operational correspondence up to structural congruence, we can discount any rules employing that.

- Assume $P = a(x).Q$. Then $\mu = a(b)$, we say that $u = [P][Q[x := b]]$, and $E(\mathsf{lcopy}(\mathbf{H}') \vdash P', P') = a(b)[[P][Q[x := b]]].Q[x := b_{[[P][Q[x:=b]]]}]$. We see that, $\mathbf{H}' \vdash P' = (\emptyset, \{(a(b), [P][Q[x := b]], [P][Q[x := b]])\}, \emptyset) \vdash Q[x := b]$, and by Lemma 4.4 the result follows.

- Assume $P = \overline{a}(b).Q$. This case is similar to the previous.

- Assume $P = P_0|P_1$. Then $E(\mathsf{lcopy}(\mathbf{H}) \vdash P, P) = Q_0 \mid Q_1$ and either $Q_i \xrightarrow{\mu[m]} Q_i'$ and $Q = Q_i'|Q_{1-i}$, or $Q_0 \xrightarrow{\alpha_0[m]} Q_0'$ and $Q_1 \xrightarrow{\alpha_1[m]} Q_1'$ and $\alpha_i = a(b)$ and $\alpha_{1-i} = \overline{a}(b)$ and $\mu = \tau$ and $Q = (\nu b)(Q_0'|Q_1')$.

  If $Q_i \xrightarrow{\mu[m]} Q_i'$ and $Q = Q_i'|Q_{1-i}$ then by induction there exists u_i such that $Q_i \xrightarrow[u_i]{\mu} P_i''$ and there exists $\mathbf{H}_i' \vdash P_i'$, such that $[\check{i}]\mathbf{H} \vdash P_i \overset{\mu}{\underset{u_i}{\rightsquigarrow}} \mathbf{H}_i' \vdash P_i'$ and $P_i'' \equiv E(\mathsf{lcopy}(\mathbf{H}_i') \vdash P_i', P_i')$. We therefore get $(\emptyset, \emptyset, \emptyset) \vdash P \xrightarrow[iu_i]{\mu} (i\overline{H_i}, i\underline{H_i}, iH_i) \vdash P_0'|P_1'$ with $P_{1-i}' = P_{1-i}$.

  If $Q_0 \xrightarrow{\alpha_0[m]} Q_0'$ and $Q_1 \xrightarrow{\alpha_1[m]} Q_1'$ and $\alpha_i = a(b)$ and $\alpha_{1-i} = \overline{a}(b)$ and $\mu = \tau$ and $Q = (\nu b)(Q_0'|Q_1')$, then by induction there exist u_i such that $Q_i \xrightarrow[u_i]{\alpha_i} P_i''$ and there exists $\mathbf{H}_i' \vdash P_i'$, such that $[\check{i}]\mathbf{H} \vdash P_i \overset{\mu}{\underset{u_i}{\rightsquigarrow}} \mathbf{H}_i' \vdash P_i'$ and $P_i'' \equiv E(\mathsf{lcopy}(\mathbf{H}_i') \vdash P_i', P_i')$ for $i \in \{0, 1\}$. We say $u = \langle u_0, u_1 \rangle$ and get $(\emptyset, \emptyset, \emptyset) \vdash P \xrightarrow[u]{\mu} (\emptyset, \emptyset, ((\alpha_0, \alpha_1), u, u) \vdash (\nu b)(P_0''|P_1'')$

- Assume $P = (\nu x)Q$. Then $x \notin n(\mu)$, $E(\mathsf{lcopy}(\mathbf{H}') \vdash P', P') = (\nu x)Q'$ and $Q \xrightarrow{\mu[u]} Q'$. We therefore get $P' = (\nu x)Q''$, and by induction $\mathbf{H} \vdash Q \xrightarrow[u]{\mu} \mathbf{H}' \vdash Q'$, and therefore $\mathbf{H} \vdash P \xrightarrow[u]{\mu} \mathbf{H}' \vdash P'$.

- Assume $P = !Q$. Then the transition must involve structural congruence and $!Q|Q \xrightarrow{\mu[u]} P'''$ for $P''' \equiv P''$, and the rest follows from the parallel case.

If for any $(\mu', u') \in \overline{H} \cup \underline{H} \cup H$, if there exists a key m, such that $E(\mathbf{H} - (\mu', u') \vdash P, P) \xrightarrow{\mu[m]} E(\mathbf{H}'' \vdash P', P')$, then there exists a location u such that $\mathbf{H} - (\mu', u') \vdash P \xrightarrow[u]{\mu} \mathbf{H}'' \vdash P'$, then having more past extrusions does not stop $\mathbf{H} - (\mu', u') \vdash P$ from performing any forwards actions and having more past actions does not allow $E(\mathbf{H} - (\mu', u') \vdash P, P)$ to perform additional forward actions.

We then need to prove that if there exists a location u such that $\mathbf{H} \vdash P \overset{\mu}{\underset{u}{\rightsquigarrow}} \mathbf{H}' \vdash P'$, then there exists a key m, such that $E(\mathsf{lcopy}(\mathbf{H}) \vdash P, P) \overset{\mu[m]}{\rightsquigarrow} E(\mathsf{lcopy}(\mathbf{H}') \vdash P', P')$.

This follows naturally from the above properties, and Propositions 3.5 and 2.6.

We finally need to prove that if $E(\mathsf{lcopy}(\mathbf{H}) \vdash P, P) \overset{\mu[n]}{\rightsquigarrow} Q$ then there exists a location, u, such that $E(\mathsf{lcopy}(\mathbf{H}) \vdash P, P) \overset{\mu[u]}{\rightsquigarrow} P''$, and there exists a πIH process, $\mathbf{H}' \vdash P'$, such that $\mathbf{H} \vdash P \overset{\mu}{\underset{u}{\rightsquigarrow}} \mathbf{H}' \vdash P'$ and $P'' \equiv E(\mathsf{lcopy}(\mathbf{H}') \vdash P', P')$.

As we have proven the above properties, and Propositions 3.5, and 2.6, we only need to prove that there exists a πIH-calculus process $\mathbf{H}' \vdash P'$, such that $P'' \equiv E(\mathsf{lcopy}(\mathbf{H}') \vdash P', P')$. Since none of the transition rules - forward or reverse - in the πIK-calculus can create unguarded choice from guarded choice, and E only generates πI-calculus processes with guarded choice, we know P'' has guarded choice.

If P'' is a standard process, then $\mathbf{H}' \vdash P' = (\emptyset, \emptyset, \emptyset) \vdash P''$. Otherwise, by Theorems 3.9 and 2.14, P'' must be forwards reachable from a standard process P''' such that $P''' \equiv E((\emptyset, \emptyset, \emptyset) \vdash P''', P''')$, and by the above properties, $\mathbf{H}' \vdash P'$ exists.

E. Proofs from Section 6

E.1. Proof of Proposition 6.3

Lemma E.1. *Let P be a consistent process and $\mathcal{N} \supseteq n(P)$ be a set of names such that $[\![P]\!]_{(\mathcal{N},l)} = \langle \mathcal{E}_0, \mathsf{Init}_0, k_0 \rangle$, $[\![P]\!]_{(\mathcal{N},l+1)} = \langle \mathcal{E}_1, \mathsf{Init}_1, k_1 \rangle$, and $\mathcal{E}_0 \leq \mathcal{E}_1$. Then for any process P', where $[\![P']\!]_{(\mathcal{N},l)} = [\![P']\!]_{(\mathcal{N},l)+1}$ and $n(P') \subseteq \mathcal{N}$, if $[\![P + P']\!]_{(\mathcal{N},l)} = \left\langle \mathcal{E}'_0, \mathsf{Init}'_0, k'_0 \right\rangle$ and $[\![P + P']\!]_{(\mathcal{N},l+1)} = \left\langle \mathcal{E}'_1, \mathsf{Init}'_1, k'_1 \right\rangle$, then $\mathcal{E}'_0 \leq \mathcal{E}'_1$.*

PROOF. Obvious.

Lemma E.2. *Let P be a consistent process and $\mathcal{N} \supseteq n(P)$ be a set of names such that $[\![P]\!]_{(\mathcal{N},l)} = \langle \mathcal{E}_0, \mathsf{Init}_0, k_0 \rangle$, $[\![P]\!]_{(\mathcal{N},l+1)} = \langle \mathcal{E}_1, \mathsf{Init}_1, k_1 \rangle$, and $\mathcal{E}_0 \leq \mathcal{E}_1$. Then for any action α, if $[\![\alpha.P]\!]_{(\mathcal{N},l)} = \left\langle \mathcal{E}'_0, \mathsf{Init}'_0, k'_0 \right\rangle$ and $[\![\alpha.P]\!]_{(\mathcal{N},l+1)} = \left\langle \mathcal{E}'_1, \mathsf{Init}'_1, k'_1 \right\rangle$, then $\mathcal{E}'_0 \leq \mathcal{E}'_1$.*

PROOF. Obvious.

Lemma E.3. *Let P be a consistent process and $\mathcal{N} \supseteq n(P)$ be a set of names such that $[\![P]\!]_{(\mathcal{N},l)} = \langle \mathcal{E}_0, \mathsf{Init}_0, k_0 \rangle$, $[\![P]\!]_{(\mathcal{N},l+1)} = \langle \mathcal{E}_1, \mathsf{Init}_1, k_1 \rangle$, and $\mathcal{E}_0 \leq \mathcal{E}_1$. Then for any process P' where $n(P') \subseteq \mathcal{N}$, if $[\![P | P']\!]_{(\mathcal{N},l)} = \left\langle \mathcal{E}'_0, \mathsf{Init}'_0, k'_0 \right\rangle$ and $[\![P | P']\!]_{(\mathcal{N},l+1)} = \left\langle \mathcal{E}'_1, \mathsf{Init}'_1, k'_1 \right\rangle$, then $\mathcal{E}'_0 \leq \mathcal{E}'_1$.*

PROOF. Let $[\![P']\!] = \left\langle \mathcal{E}', \mathsf{Init}', k' \right\rangle$. Obviously $\mathcal{E}_0 \times \mathcal{E}_2 \leq \mathcal{E}_1 \times \mathcal{E}_2$ and for any $(e, *), (*, e') \in E'_0$, there exist names $x \in no(\lambda'_0((e, *)))$ and a such that $\lambda'_0((*, e')) = \overline{a}(x)$ if and only if $x \in no(\lambda'_1((e, *)))$ and $\lambda'_1((*, e')) = \overline{a}(x)$. Additionally, the standard bound names of P_0 and P_1. are the same.

Lemma E.4. *Let P_0 and P_1 be consistent processes and $\mathcal{N} \supseteq n(P_0) \cup n(P_1)$ be a set of names such that $[\![P_0]\!]_{(\mathcal{N},l)} = \langle \mathcal{E}_0, \mathsf{Init}_0, k_0 \rangle$, $[\![P_1]\!]_{(\mathcal{N},l)} = \langle \mathcal{E}_1, \mathsf{Init}_1, k_1 \rangle$, and $\mathcal{E}_0 \leq \mathcal{E}_1$. Then for any name n, if $[\![(\nu n) P_0]\!]_{(\mathcal{N},l)} = \left\langle \mathcal{E}'_0, \mathsf{Init}'_0, k'_0 \right\rangle$ and $[\![(\nu n) P_1]\!]_{(\mathcal{N},l)} = \left\langle \mathcal{E}'_1, \mathsf{Init}'_1, k'_1 \right\rangle$, then $\mathcal{E}'_0 \leq \mathcal{E}'_1$.*

PROOF. Obvious.

E.2. Proof of Proposition 6.5

PROOF. We say that $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$ and $\mathcal{E}' = (E', F', \mapsto', \sharp', \rhd', \lambda', \mathsf{Act}')$ and do a case analysis on the structural congruence rules:

$P = P_0|P_1$ **and** $P' = P_1|P_0$**:** In this case, products are unique up to isomorphism, and the function $f(e) = \begin{cases} (e_1, e_0) & \text{if } e = (e_0, e_1) \\ (e_1, *) & \text{if } e = (*, e_1) \\ (*, e_0) & \text{if } e = (e_0, *) \end{cases}$ clearly fulfils the other conditions and remains a morphism after the enablings and preventions describing the link dependencies are added to the product.

$P = P_0|(P_1|P_2)$ **and** $P' = (P_0|P_1)|P_2$**:** We know that products are associative up to isomorphism, and the function $f((e_0, (e_1, e_2)) = ((e_0, e_1), e_2)$ clearly fulfils the other conditions and remains a morphism after the enablings and preventions describing the link dependencies are added to the product.

$P = P'|0$**:** If $f((e, *)) = e$, then this clearly holds.

$P = P_0 + P_1$ **and** $P' = P_1 + P_0$**:** Coproducts are unique up to isomorphism, and the mapping $f(i, e) = (1 - i, e)$ clearly fulfils the other conditions.

$P = P_0 + (P_1 + P_2)$ **and** $P' = (P_0 + P_1) + P_2$**:** We know that coproducts are associative up to isomorphism, and $f((e_0, (e_1, e_2))) = ((e_0, e_1), e_2)$ clearly fulfils the other conditions.

$P = P' + 0$**:** Clearly $f(0, e) = e$ is an isomorphism, $\mathsf{Init} = \{0\} \times \mathsf{Init}'$, and $k(0, e) = k'(e)$.

$P = !_{\mathbf{x}}Q$**,** $P' = !_{\mathbf{x}\setminus\{x_0,\ldots,x_k\}}Q|Q\{^{x_0,\ldots,x_k}/_{a_0,\ldots,a_k}\}$ **,** $\{x_0, \ldots, x_k\} \subseteq \mathbf{x}$**,** and $\mathsf{bn}(P) = \{a_0, \ldots, a_k\}$**:** Straightforward from the requirement that all free and bound names are distinct.

E.3. Proof of Theorem 6.7

PROOF. Let $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$ and $\mathcal{E}' = (E', F', \mapsto', cf', \rhd', \lambda', \mathsf{Act}')$. We prove the theorem by induction on $P \xrightarrow{\mu[m]} P'$:

1. Suppose $P = a(x).Q$, $P' = a(x)[m].Q[x := b_{[n]}]$, $\mathsf{std}(Q)$, and $\mu = a(b)$. Then for all $n \in (\mathcal{N} \setminus \mathsf{sbn}(Q)) = (\mathcal{N} \setminus \mathsf{sbn}(Q[x := b_{[n]}]))$, we have $\{\!|Q[x := n]|\!\} = \langle \mathcal{E}_n, \mathsf{Init}_n, k_n \rangle$, $\left\{\!\left|Q[x := b_{[n]}][b_{[n]} := n]\right|\!\right\} = \left\langle \mathcal{E}'_n, \mathsf{Init}'_n, k'_n \right\rangle$, and an isomorphism $f_n : \mathcal{E}_n \to \mathcal{E}'_n$. We define our isomorphism

$$f((n, e_n)) = \begin{cases} (n, f_n(e_n)) & \text{if } e_n \in E_n \\ (n, e'_n) & \text{for } \{e'_n\} = \{e' \mid (n, e') \in E' \text{ and } e' \notin E'_n\} \text{ otherwise} \end{cases}$$

Since all bound names are different from all other bound and free names, $b \notin \mathsf{bn}(Q)$, and therefore there exists an $e \in E$ such that $\lambda(e) = a(b)$, and for all $e' \in E$ either $e' = e$, $e' \sharp e$, or $\{e\} \mapsto e'$. We therefore get $\mathsf{Init} = \emptyset \xrightarrow{\{e\}} X$ and $f(X) = \mathsf{Init}'$, and the rest of the conditions fulfilled.

2. Suppose $P = \overline{a}(x).Q$, $P' = \overline{a}(x)[m].Q$, $\mu = \overline{a}(x)$, and $\mathsf{std}(Q)$. This case is similar to the previous, without the choice of substitutions.

3. Suppose $P = \alpha[n].Q$, $P' = \alpha[n].Q'$, $Q \xrightarrow{\mu[m]} Q'$, $m \neq n$, and if $\mu = \overline{a}(x)$ then $x \notin n(\alpha)$. Then let $\{\!|Q|\!\} = \left\langle \mathcal{E}_Q, \mathsf{Init}_Q, k_Q \right\rangle$ and $\{\!|Q'|\!\} = \left\langle \mathcal{E}'_Q, \mathsf{Init}'_Q, k'_Q \right\rangle$. We have an isomorphism $f_Q : \mathcal{E}_Q \to \mathcal{E}'_Q$ and a transition $\mathsf{Init}_Q \xrightarrow{e_Q} X_Q$ such that $\lambda_Q(e_Q) = \mu$, $f \circ k'_Q = k_Q[e_Q \mapsto m]$, and $f(X_Q) = \mathsf{Init}'_Q$. We define our isomorphism

$$f((n, e_n)) = \begin{cases} (n, f_n(e_n)) & \text{if } e_n \in E_n \\ (n, e'_n) & \text{for } \{e'_n\} = \{e' \mid (n, e') \in E' \text{ and } e' \notin E'_n\} \text{ otherwise} \end{cases}$$

and $e = (x, e_Q)$ if $\alpha = a(x)$, and

$$f(e') = \begin{cases} f_Q(e') & \text{if } e' \in E_Q \\ e'' & \text{for } \{e''\} = \{e''' \mid e''' \in E' \text{ and } e''' \notin E'_Q\} \text{ otherwise} \end{cases}$$

and $e = e_Q$ if $\alpha = \overline{a}(x)$. These clearly fulfil the conditions.

4. Suppose $P = P_0|P_1$, $P' = P'_0|P_1$, $P_0 \xrightarrow{\mu[m]} P'_0$, $\mathsf{fsh}[n](P_1)$, and if $\mu = \overline{a}(x)$ then $x \notin \mathsf{fn}(P_1)$. Then let $\{\!|P_0|\!\} = \langle \mathcal{E}_0, \mathsf{Init}_0, k_0 \rangle$, $\left\{\!\left|P'_0\right|\!\right\} = \left\langle \mathcal{E}'_0, \mathsf{Init}'_0, k'_0 \right\rangle$, and $\{\!|P_1|\!\} = \langle \mathcal{E}_1, \mathsf{Init}_1, k_1 \rangle$. We then have an isomorphism $f_0 : \mathcal{E}_0 \to \mathcal{E}'_0$ and transition $\mathsf{Init}_0 \xrightarrow{e_0} X_0$ such that $\lambda_0(e_0) = \mu$, $f_0 \circ k'_0 = k_0[e_0 \mapsto m]$, and $f_0(X_0) = \mathsf{Init}'_0$. We define our isomorphism

$$f(e') = \begin{cases} (f_0(e'_0), *) & \text{if } e' = (e'_0, *) \\ (*, e'_1) & \text{if } e' = (*, e'_1) \\ (f_0(e'_0), e'_1) & \text{if } e' = (e'_0, e'_1) \end{cases}$$

and $e = (e_0, *)$. Since $\mathsf{sbn}(P_0) = \mathsf{sbn}(P'_0)$ this is an isomorphism, and since all free and bound names are different, $no(\mu) \cap \mathsf{sbn}(P_1) = \emptyset$, implying $\mathsf{Init} \xrightarrow{e}$. The other conditions are clearly fulfilled.

5. Suppose $P = P_0|P_1$, $P' = (\nu x)(P'_0|P_1)$, $\mu = \tau$, $P_0 \xrightarrow{a(x)[m]} P'_0$, and $P_1 \xrightarrow{\overline{a}(x)[m]} P'_1$. Then let $\{\!|P_0|\!\} = \langle \mathcal{E}_0, \mathsf{Init}_0, k_0 \rangle$, $\left\{\!\left|P'_0\right|\!\right\} = \left\langle \mathcal{E}'_0, \mathsf{Init}'_0, k'_0 \right\rangle$, $\{\!|P_1|\!\} = \langle \mathcal{E}_1, \mathsf{Init}_1, k_1 \rangle$, and $\left\{\!\left|P'_1\right|\!\right\} = \left\langle \mathcal{E}'_1, \mathsf{Init}'_1, k'_1 \right\rangle$. Then we have isomorphisms $f_0 : \mathcal{E}_0 \to \mathcal{E}'_0$ and $f_1 : \mathcal{E}_1 \to \mathcal{E}'_1$ and transitions $\mathsf{Init}_0 \xrightarrow{e_0} X_0$ and $\mathsf{Init}_1 \xrightarrow{e_1} X_1$ such that $\lambda_0(e_0) = a(x)$, $\lambda_1(e_1) = \overline{a}(x)$, $f_0 \circ k'_0 = k_0[e_0 \mapsto m]$, $f_1 \circ k'_1 = k_1[e_1 \mapsto m]$, $f_0(X_0) = \mathsf{Init}'_0$, and $f_1(X_1) = \mathsf{Init}'_1$. We then define our isomorphism

$$f(e') = \begin{cases} (f_0(e'_0), *) & \text{if } e' = (e'_0, *) \\ (*, f_1(e'_1)) & \text{if } e' = (*, e'_1) \\ (f_0(e'_0), f_1(e'_1)) & \text{if } e' = (e'_0, e'_1) \end{cases}$$

and $e = (e_0, e_1)$. Since $\mathsf{sbn}(P_0) = \mathsf{sbn}(P_0')$ and the existence of $(f_0(e_0), f_1(e_1)) \in \mathsf{Init}'$ and $a(x)[m]$ and $\overline{a}(x)[m]$ prevents (νx) from affecting $\mathcal{E}'$, f is an isomorphism, and since $no(\tau) = \emptyset$, we have a transition $\mathsf{Init} \xrightarrow{e}$. The other conditions are clearly fulfilled.

6. Suppose $P = P_0 + P_1$, $P' = P_0' + P_1$, $P_0 \xrightarrow{\mu[m]} P_0'$, and $\mathsf{std}(P_1)$. Then let $\{\!|P_0|\!\} = \langle \mathcal{E}_0, \mathsf{Init}_0, k_0 \rangle$, $\{\!|P_0'|\!\} = \langle \mathcal{E}_0', \mathsf{Init}_0', k_0' \rangle$, and $\{\!|P_1|\!\} = \langle \mathcal{E}_1, \mathsf{Init}_1, k_1 \rangle$. We then have an isomorphism $f_0 : \mathcal{E}_0 \to \mathcal{E}_0'$ and transition $\mathsf{Init}_0 \xrightarrow{e_0} X_0$ such that $\lambda_0(e_0) = \mu$, $f_0 \circ k_0' = k_0[e_0 \mapsto m]$, and $f_0(X_0) = \mathsf{Init}_0'$. We define out isomorphism
$$f((i, e_i)) = \begin{cases} (0, f_0(e_0)) & \text{if i=0} \\ (1, e_1) & \text{if i=1} \end{cases}$$
and $e = (0, e_0)$. Isomorphism is preserved by the coproduct, and the remaining conditions are clearly fulfilled.

7. Suppose $P = (\nu x)Q$, $P' = (\nu x)Q'$, $Q \xrightarrow{\mu[m]} Q'$, and $x \notin n(\mu)$. Then let $\{\!|Q|\!\} = \langle \mathcal{E}_Q, \mathsf{Init}_Q, k_Q \rangle$ and $\{\!|Q'|\!\} = \left\langle \mathcal{E}_Q', \mathsf{Init}_Q', k_Q' \right\rangle$. We have an isomorphism $f_Q : \mathcal{E}_Q \to \mathcal{E}_Q'$ and a transition $\mathsf{Init}_Q \xrightarrow{e_Q} X_Q$ such that $\lambda_Q(e_Q) = \mu$, $f \circ k_Q' = k_Q[e_Q \mapsto m]$, and $f(X_Q) = \mathsf{Init}_Q'$. Either there exist past actions $b(a)[m]$ and $\overline{b}(a)[m]$ in P which are not guarded by a restriction (νa) in P or not. If such $b(a)[m]$ and $\overline{b}(a)[m]$ exist, then $\langle \mathcal{E}, \mathsf{Init}, k \rangle = \langle \mathcal{E}_Q, \mathsf{Init}_Q, k_Q \rangle$ and $\langle \mathcal{E}', \mathsf{Init}', k' \rangle = \left\langle \mathcal{E}_Q', \mathsf{Init}_Q', k_Q' \right\rangle$, and the rest follows trivially. Otherwise restriction preserves morphisms, and clearly does not affect $e = e_Q$.

8. Suppose $P \equiv Q$, $P' \equiv Q'$, and $Q \xrightarrow{\mu[m]} Q'$. Then the result follows from induction and Proposition 6.5

E.4. Proof of Theorem 6.8

PROOF. We prove this by structural induction on P:

- Suppose $P = 0$. Then $E = \emptyset$, and no transition $\mathsf{Init} \xrightarrow{\{e\}} X$ exists.
- Suppose $P = \overline{a}(x).Q$. Let $\{\!|Q|\!\}_{\mathcal{N}} = \langle \mathcal{E}_Q, \mathsf{Init}_Q, k_Q \rangle$, $\mathcal{E}_Q = (E_Q, F_Q, \mapsto_Q, \sharp_Q, \rhd_Q, \lambda_Q, \mathsf{Act}_Q)$, and $\mathcal{E}' = (E', F', \mapsto', \sharp', \rhd', \lambda', \mathsf{Act}')$. Then there exists e such that $E \setminus E_Q = \{e\}$, and for all $e' \in E$, if $e' \neq e$ then $\{e\} \mapsto e'$. Therefore this is the only possible e such that $\mathsf{Init} \xrightarrow{\{e\}}$. Additionally we have $\lambda(e) = \overline{a}(x)$ and $P \xrightarrow{\overline{a}(x)[m]} \overline{a}(x)[m].Q$ for any key m, and the rest of the case is straightforward.
- Suppose $P = a(x).Q$. Then there must exist $b \in \mathcal{N} \setminus \mathsf{sbn}(P)$ such that $\lambda(e) = a(b)$, and for all $e' \in E$ either $e = e'$, $e \sharp e'$, or $\{e\} \mapsto e'$. There then exists a transition $P \xrightarrow{a(b)[m]} a(b)[m].Q[x := b_{[m]}]$ and the rest of the case is straightforward.

- Suppose $P = \overline{a}(x)[n].Q$. Let $\{\!|Q|\!\}_{\mathcal{N}} = \left\langle \mathcal{E}_Q, \mathsf{Init}_Q, k_Q \right\rangle$, $\mathcal{E}_Q = (E_Q, F_Q, \mapsto_Q, \sharp_Q, \rhd_Q, \lambda_Q, \mathsf{Act}_Q)$, and $\mathcal{E}' = (E', F', \mapsto', \sharp', \rhd', \lambda', \mathsf{Act}')$. Then $\mathsf{Init} \xrightarrow{\{e\}} X$ implies $\mathsf{Init}_Q \xrightarrow{\{e\}} X \cap E_Q$. We therefore have a transition $Q \xrightarrow{\mu[m]} Q'$ such that $\{\!|Q'|\!\}_{\mathcal{N}} = \left\langle \mathcal{E}'_Q, \mathsf{Init}'_Q, k'_Q \right\rangle$ and an isomorphism $f_Q : \mathcal{E}_Q \to \mathcal{E}'_Q$ such that $\lambda_Q(e) = \mu$, $f_Q \circ k'_Q = k_Q[e \mapsto m]$, and $f_Q(X \cap E_Q) = \mathsf{Init}'_Q$. This gives us a transition $P \xrightarrow{\mu[m]} \overline{a}(x)[n].Q'$ and the rest of the case is straightforward.

- Suppose $P = a(x)[n].Q$. This case is a combination of the previous two.

- Suppose $P = Q + R$. Let $\{\!|Q|\!\}_{\mathcal{N}} = \left\langle \mathcal{E}_Q, \mathsf{Init}_Q, k_Q \right\rangle$, $\mathcal{E}_Q = (E_Q, F_Q, \mapsto_Q, \sharp_Q, \rhd_Q, \lambda_Q, \mathsf{Act}_Q)$, $\{\!|R|\!\}_{\mathcal{N}} = \langle \mathcal{E}_R, \mathsf{Init}_R, k_R \rangle$, $\mathcal{E}_R = (E_R, F_R, \mapsto_R, \sharp_R, \rhd_R, \lambda_R, \mathsf{Act}_R)$, and $\mathcal{E}' = (E', F', \mapsto', \sharp', \rhd', \lambda', \mathsf{Act}')$. Either $e = (0, e_Q)$ or $e = (1, e_R)$. In the first case we get a transition transition $Q \xrightarrow{\mu[m]} Q'$ such that $\{\!|Q'|\!\}_{\mathcal{N}} = \left\langle \mathcal{E}'_Q, \mathsf{Init}'_Q, k'_Q \right\rangle$ and an isomorphism $f_Q : \mathcal{E}_Q \to \mathcal{E}'_Q$ such that $\lambda_Q(e_Q) = \mu$, $f_Q \circ k'_Q = k_Q[e_Q \mapsto m]$, and $f_Q(\{e'_Q | (0, e'_Q) \in X\}) = \mathsf{Init}'_Q$. We therefore define

$$f(e) = \begin{cases} (0, f_Q(e_Q)) & \text{if } e = (0, e_Q) \\ e & \text{otherwise} \end{cases}$$

and the rest of the case is straightforward. If $e = (1, e_R)$, the proof is similar.

- Suppose $P = Q|R$. Let $\{\!|Q|\!\}_{\mathcal{N}} = \left\langle \mathcal{E}_Q, \mathsf{Init}_Q, k_Q \right\rangle$, $\{\!|R|\!\}_{\mathcal{N}} = \langle \mathcal{E}_R, \mathsf{Init}_R, k_R \rangle$, $\mathcal{E}_Q = (E_Q, F_Q, \mapsto_Q, \sharp_Q, \rhd_Q, \lambda_Q, \mathsf{Act}_Q)$, $\mathcal{E}_R = (E_R, F_R, \mapsto_R, \sharp_R, \rhd_R, \lambda_R, \mathsf{Act}_R)$, and $\mathcal{E}' = (E', F', \mapsto', \sharp', \rhd', \lambda', \mathsf{Act}')$. Either $e = (e_Q, *)$, $e = (*, e_R)$, or $e = (e_Q, e_R)$. If $e = (e_Q, *)$ then we have a transition $Q \xrightarrow{\mu[m]} Q'$ such that $\{\!|Q'|\!\}_{\mathcal{N}} = \left\langle \mathcal{E}'_Q, \mathsf{Init}'_Q, k'_Q \right\rangle$ and an isomorphism $f_Q : \mathcal{E}_Q \to \mathcal{E}'_Q$ such that $\lambda_Q(e_Q) = \mu$, $f_Q \circ k'_Q = k_Q[e_Q \mapsto m]$, and $f_Q(\{e'_Q | (e'_Q, *) \in X \text{ or } (e'_Q, e'_R) \in X\}) = \mathsf{Init}'_Q$. We therefore get $P \xrightarrow{\mu[m]} Q'|R$ so long as $\mathsf{fsh}[m](R)$, and if not we can do the same with a different m. We can then define

$$f(e') = \begin{cases} (f_0(e'_0), *) & \text{if } e' = (e'_0, *) \\ (*, e'_1) & \text{if } e' = (*, e'_1) \\ (f_0(e'_0), e'_1) & \text{if } e' = (e'_0, e'_1) \end{cases}$$

and the rest of the case is straightforward. If $e = (*, e_R)$, the case is similar. If $e = (e_Q, e_R)$, then we have transition $Q \xrightarrow{\alpha[m]} Q'$ such that $\{\!|Q'|\!\}_{\mathcal{N}} = \left\langle \mathcal{E}'_Q, \mathsf{Init}'_Q, k'_Q \right\rangle$ and isomorphism $f_Q : \mathcal{E}_Q \to \mathcal{E}'_Q$ such that $\lambda_Q(e_Q) = \alpha$, $f_Q \circ k'_Q = k_Q[e_Q \mapsto m]$, and $f_Q(\{e'_Q | (e'_Q, *) \in X \text{ or } (e'_Q, e'_R) \in X\}) = \mathsf{Init}'_Q$, and transition $R \xrightarrow{\alpha'[m]} R'$ such that $\{\!|R'|\!\}_{\mathcal{N}} = \left\langle \mathcal{E}'_R, \mathsf{Init}'_R, k'_R \right\rangle$ and isomorphism $f_R : \mathcal{E}_R \to \mathcal{E}'_R$ such that $\lambda_R(e_R) = \alpha'$, $f_R \circ k'_R = k_R[e_R \mapsto m]$, and $f_R(\{e'_R | (*, e'_R) \in X \text{ or } (e'_Q, e'_R) \in$

$X\}) = \mathsf{Init}'_R$ and there exist names a, b such that either $\alpha = a(b)$ and $\alpha' = \overline{a}(b)$ or $\alpha' = a(b)$ and $\alpha = \overline{a}(b)$. We therefore get a transition $P \xrightarrow{\tau[m]} (\nu b)(Q' \mid R')$ and define

$$f(e) = \begin{cases} (f_0(e'_0), *) & \text{if } e = (e'_0, *) \\ (*, f_1(e'_1)) & \text{if } e = (*, e'_1) \\ (f_0(e'_0), f_1(e'_1)) & \text{if } e = (e'_0, e'_1) \end{cases}$$

and the rest of the case is straightforward.

- Suppose $P = (\nu a)Q$. Let $\{\!|Q|\!\}_{\mathcal{N}} = \left\langle \mathcal{E}_Q, \mathsf{Init}_Q, k_Q \right\rangle$, $\mathcal{E}_Q = (E_Q, F_Q, \mapsto_Q, \sharp_Q, \rhd_Q, \lambda_Q, \mathsf{Act}_Q)$, and $\mathcal{E}' = (E', F', \mapsto', \sharp', \rhd', \lambda', \mathsf{Act}')$. Then either there exist past actions $b(a)[m]$ and $\overline{b}(a)[m]$ in Q which are not guarded by a restriction (νa) in Q or not. If such $b(a)[m]$ and $\overline{b}(a)[m]$ do exist, then they must be in parallel, and therefore there exists an event $e' \in E \setminus \mathsf{Init}$ such that $\lambda(e) = \overline{b}(a)$, and for all other events $e'' \in E$, if $\lambda(e')$ outputs a then $e' = e$, and if $a \in no(\lambda(e'))$ then $\{e'\} \mapsto e''$. Additionally there exists $e''' \in \mathsf{Init}$ such that $e''' \sharp e'$ and $\lambda(e''') = \tau$. We therefore get that $a \notin n(e)$. Additionally $\mathsf{Init}_Q = \mathsf{Init} \xrightarrow{\{e\}} X$ and by induction we have a transition $Q \xrightarrow{\mu[m]} Q'$ such that $\{\!|Q'|\!\}_{\mathcal{N}} = \left\langle \mathcal{E}'_Q, \mathsf{Init}'_Q, k'_Q \right\rangle$ and an isomorphism $f_Q : \mathcal{E}_Q \to \mathcal{E}'_Q$ such that $\lambda_Q(e) = \mu$, $f_Q \circ k'_Q = k_Q[e \mapsto m]$, and $f_Q(X) = \mathsf{Init}'_Q = \mathsf{Init}'$. We define $f = f_Q$ and the result follows. If no such $b(a)[m]$ and $\overline{b}(a)[m]$ exist in Q then clearly $a \notin n(\lambda(e))$, and restriction preserves morphisms, meaning the proof is straightforward.

F. Proofs from Section 7

F.1. Proof of Theorem 7.3

PROOF. I is clearly irreflexive and symmetric, based on Definition 2.9.

We prove the three requirements on the definition of an LATS:

1. If $(\alpha, u, D) \in E$, then there exist processes $\mathbf{H} \vdash P$ and $\mathbf{H}' \vdash P'$ such that $\mathbf{H} \vdash P \xrightarrow[u,D]{\alpha} \mathbf{H}' \vdash P'$ by definition.
2. If $\mathbf{H} \vdash P \xrightarrow[u,D]{\alpha} \mathbf{H}' \vdash P'$ and $\mathbf{H} \vdash P \xrightarrow[u,D]{\alpha} \mathbf{H}'' \vdash P''$ then we can use similar arguments to Theorem 4.2 of [14] to show that $P' \equiv P''$ and $\mathbf{H}' = \mathbf{H}''$. Unlike [14], we do not need to take into account a (Rep) rule, eliminating that case, but instead a [STR] rule, which we do by only requiring $P' \equiv P''$, rather than $P' = P''$.
3. If $\mathbf{H} \vdash P \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_0 \vdash P_0$, $\mathbf{H} \vdash P \xrightarrow[u_1,D_1]{\alpha_1} \mathbf{H}_1 \vdash P_1$ and (α_0, u_0, D_0) and (α_1, u_1, D_1) are independent then by Proposition 2.10, we get $\mathbf{H}_0 \vdash P_0 \xrightarrow[u_1,D_1]{\alpha_1} \mathbf{H}_2 \vdash P_2$ and $\mathbf{H}_1 \vdash P_1 \xrightarrow[u_0,D_0]{\alpha_0} \mathbf{H}_2 \vdash P_2$ for some $\mathbf{H}_2 \vdash P_2$.

4. If $\mathbf{H} \vdash P \xrightarrow[u_0, D_0]{\alpha_0} \mathbf{H}_0 \vdash P_0$, $\mathbf{H}_0 \vdash P_0 \xrightarrow[u_1, D_1]{\alpha_1} \mathbf{H}_2 \vdash P_2$ and (α_0, u_0, D_0) and (α_1, u_1, D_1) are independent then we can use similar arguments to Theorem 4.2 of [14] to show that $\mathbf{H} \vdash P \xrightarrow[u_1, D_1]{\alpha_1} \mathbf{H}_1 \vdash P_1$ and $\mathbf{H}_1 \vdash P_1 \xrightarrow[u_0, D_0]{\alpha_0} \mathbf{H}_2 \vdash P_2$ for some $\mathbf{H}_1 \vdash P_1$.

G. Proofs from Section 8

G.1. Proof of Lemma 8.2

PROOF. We first prove that if X is a possible cause, then it fulfils these conditions by induction on the size of X.

If $X = \emptyset$, then this is obvious.

Otherwise, if $X \neq \emptyset$ is a possible cause of e then for each $e' \in X$, these exists $X' \subseteq (X \setminus \{e'\}$ such that X' is a possible cause of e'. By definition, X is finite and conflict free. Additionally, there can only exist one such X', since different possible causes of the same event must include different events from the same bundle, and are therefore in conflict. We can therefore say that $e' <_X e''$ if there exists a possible cause of e'', X'', such that $X' \subseteq X'' \subseteq X$, and since events cannot appear in their own possible causes, $\leq_X$ is a partial order. By induction, X must then be reachable, by performing the events in a sequence ordered by $<_X$.

Since X contains one event from each bundle associated with e, and $X \cup \{e\}$ is conflict free and therefore a configuration, $X \xrightarrow{e}$.

For any reachable configuration, $X'' \subset X$, since X'' is reachable, if $e' \in X \setminus X''$, then there cannot exist $e'' \in X''$ such that $e' <_X e''$. Therefore there must exist at least one $e''' \in X \setminus X''$ such that e''' is a maximal element of X with regards to $<_X$. This means that $e''' \in Y \mapsto e$ for some Y, and therefore $X'' \cap Y = \emptyset$, and $X'' \not\xrightarrow{e}$.

If X is reachable, $X \xrightarrow{e}$, and for all reachable configurations $X'' \subset X$, $X'' \not\xrightarrow{e}$, and for any e_Y and $Y \subset X$ such that Y is reachable, $Y \xrightarrow{e_Y}$, and for all reachable configurations $Y'' \subset Y$, $Y'' \not\xrightarrow{e_Y}$, Y is a possible cause of e_Y, then we show that X fulfils the conditions of Definition 5.3:

- Since $X \xrightarrow{e}$, $e \notin X$.
- X is finite by definition.
- Since $X \xrightarrow{e}$, by definition, for any $X' \mapsto e$, $X' \cap X \neq \emptyset$.
- Since $X \xrightarrow{e}$, $X \cup \{e\}$ is a configuration and therefore conflict free.
- Since X is reachable, there exists a trace, $\emptyset \xrightarrow{e_0} X_1 \xrightarrow{e_1} \ldots X_n \xrightarrow{e_n} X$, and for any e_i, there exists $X'_i \subseteq X_i$, which fulfils the conditions and is therefore a possible cause of e_i.

- If $X' \subseteq X$ then either X' is not a reachable configuration, or $X' \overset{e}{\not\rightarrow}$. In the first case, if X' is not a configuration, then X' is not conflict-free and therefore cannot be a possible cause. If X' is a non-reachable configuration, then there exists $e' \in X'$ such that for all reachable $X'' \subset X'$, $X'' \nrightarrow e'$. As shown in the first part of this proof, this means that no $X'' \subseteq X'$ is a possible cause of e', and therefore X' is not a possible cause. If X' is a reachable configuration and $X' \overset{e}{\not\rightarrow}$, then either $X' \cup \{e\}$ is not a configuration and therefore not conflict free, or there exists $Y \mapsto e$ such that $Y \cap X' = \emptyset$. Either way, X' is not a possible cause of e.

G.2. Proof of Proposition 8.5

PROOF. We first show that given a CLRBES, $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$, $P_B(\mathcal{E})$ is a CLRPES. This is straightforward. We then show that given a morphism $f : \mathcal{E}_0 \to \mathcal{E}_1$ between two CLRBESs $P_B(f) : P_B(\mathcal{E}_0) \to P_B(\mathcal{E}_1)$ is an LRPES morphism. Let $\mathcal{E}_0 = (E_0, F_0, \mapsto_0, \sharp_0, \rhd_0, \lambda_0, \mathsf{Act}_0)$, $P_B(\mathcal{E}_0) = (E_0', F_0', <_0', \sharp_0', \prec_0', \rhd_0', \lambda_0', \mathsf{Act}_0')$, $\mathcal{E}_1 = (E_1, F_1, \mapsto_1, \sharp_1, \rhd_1, \lambda_1, \mathsf{Act}_1)$, and $P_B(\mathcal{E}_1) = (E_1', F_1', <_1', \sharp_1', \prec_0', \rhd_0', \lambda_0', \mathsf{Act}_0')$.

We then show that $P_B(f)$ fulfils the conditions of a LRPES morphism.

- Given $(X_0, e_0) \in E_0'$ with $P_B(f)((X_0, e_0)) = (X_1, e_1) \neq \bot$ then $\{(X_1', e_1') \mid (X_1', e_1') <_1' (X_1, e_1)\} = \{(X_1', e_1') \mid X_1' \cup \{e_1'\} \subseteq X_1\} \subseteq \{f((X_0', e_0')) \mid X_0' \cup \{e_0'\} \subseteq X_0\}$.

- Given $(X, e), (X', e') \in E_0'$ such that $P_B(f)((X, e)) \neq \bot \neq P_B(f)((X', e'))$, if $P_B(f)((X, e)) \; \sharp_1' \; P_B(f)((X', e'))$ then either $P_B(f)((X, e)) \; \sharp_1'' \; P_B(f)((X', e'))$, $P_B(f)((X', e')) \; \sharp_1' \; P_B(f)((X, e))$, or there exists (X'', e'') such that $(X'', e'') <_1' P_B(f)((X', e'))$ and $P_B(f)((X, e)) \; \sharp_1' \; (X'', e'')$.

  If $P_B(f)((X, e)) \; \sharp_1'' \; P_B(f)((X', e'))$ then either $f(e) \; \sharp_1 \; f(e')$ or $f(e) = f(e')$, and in either case $e \; \sharp_0 \; e'$, meaning $(X, e) \; \sharp_0' \; (X', e')$.

  If $P_B(f)((X', e')) \; \sharp_1' \; P_B(f)((X, e))$, then $(X', e') \; \sharp_0' \; (X, e)$ and since $\sharp_0'$ is closed under symmetry, $(X, e) \; \sharp_0' \; (X', e')$.

  If there exists (X'', e'') such that $(X'', e'') <_1' P_B(f)((X', e'))$ and $P_B(f)((X, e)) \; \sharp_1' \; (X'', e'')$, then $X'' \cup \{e''\} \subseteq f(X')$, meaning we have e_0'' such that $f(e_0'') = e''$ and $e_0'' \in X_0''' \mapsto_0 e'$ and $X_0'' \subseteq X'$ such that $(X'', e_0) \in E_0$ and $f(X_0'') = X''$. We therefore get $(X'', e'') <_0' (X', e')$ and $(X, e) \; \sharp_0' \; (X'', e'')$, and therefore $(X, e) \; \sharp_0' \; (X', e')$.

- $\prec_0 = \prec_0 = \emptyset \times \emptyset$.

- Given $(X, e), (X', e') \in E_0'$ such that $P_B(f)((X, e)) \neq \bot \neq P_B(f)((X', e'))$, if $P_B(f)((X, e)) \rhd_1' \underline{P_B(f)((X', e'))}$ then $P_B(f)((X', e')) <_1' P_B(f)((X, e))$, meaning $f(X') \cup \{f(e')\} \subseteq f(X)$. Since f is a morphism, we get $e \rhd_0 \underline{e'}$, and since $\mathcal{E}_0$ is causal, $e' \in X'' \mapsto_0 e$, meaning $(X', e') <_0' (X, e)$, and therefore $(X, e) \rhd_0' (X', e')$.

- If $P_B(f)((X, e)) = P_B(f)((X, e')) \neq \bot$ and $(X, e) \neq (X', e')$ then either $e = e'$ and $X \neq X'$, or $e \neq e'$ and $f(e) = f(e')$. Either way $(X, e) \; \sharp_0' \; (X', e')$.

- Straightforward.
- Straightforward.

The rest of the proof of P_B being a functor is straightforward.

G.3. Proof of Lemma 8.6

PROOF. We first prove that if $Y \xrightarrow{e}$ then there exists a reachable configuration $Y' = \{(X', e') \mid X' \cup \{e'\} \subseteq Y\}$ and $X \subseteq Y$ such that $Y' \xrightarrow{(X,e)}$. We do this by induction on the size of Y.

If $Y = \emptyset$, then clearly $\emptyset$ is a possible cause of e, and $Y' = \emptyset \xrightarrow{(\emptyset,e)}$.

If $Y \neq \emptyset$ then since $\mathcal{E}$ is causal, Y must be forwards reachable. Therefore, by induction, Y is also forwards reachable. And clearly Y' contains all causes of and no conflicts with (X, e), giving us $Y' \xrightarrow{(X,e)}$.

We then prove that if $Y' \xrightarrow{(X,e)}$ then $Y = X \cup \bigcup_{(X',e') \in Y'} X' \cup \{e'\}$ is a reachable configuration and $Y \xrightarrow{e}$. We prove this by induction on the size of Y'.

If $Y' = \emptyset$ then $X = \emptyset$ and therefore $Y = \emptyset$. And since $\emptyset$ is a possible cause of e, $Y \xrightarrow{e}$.

If $Y' \neq \emptyset$ then since $P_B(\mathcal{E})$ is causal, Y' must be forwards reachable. And since $Y' \xrightarrow{(X,e)}$, $X \subseteq Y'$, and $Y = \bigcup_{(X',e') \in Y'} X' \cup \{e'\}$. Therefore, by induction, Y is also forwards reachable. And since Y contains a possible cause of e, $Y \xrightarrow{e}$.

G.4. Proof of Theorem 8.7

PROOF. Suppose $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$, $\mathcal{E}' = (E', F', \mapsto', \sharp', \prec', \rhd', \lambda', \mathsf{Act}')$, and $P_B(\mathcal{E}) = (E_P, F_P, <_P, \sharp_P, \prec_P, \rhd_P, \lambda_P, \mathsf{Act}_P)$.

We first note that, if an event in E' has a minimal representative, t, containing an action, which either receives names not in $\mathcal{N}$, or can only be reached by using α-conversion in a way other than what is used for replication when generating $\mathcal{E}$, then e cannot cause any events, for which the above does not hold.

We prove the theorem by induction on the structure and level of nesting of replication in P.

If P does not contain replication, then we look at the structure of P:

1. Suppose $P = 0$. Then $\mathcal{E}$ and $\mathcal{E}'$ both have no events.
2. Suppose $P = a(x).Q$. Then for each name n, $\mathcal{E}'$ has an event containing the trace $(a(n), [P][Q[x := n]]), \emptyset)$ and for each name $n' \in \mathcal{N}$, $\mathcal{E}$ has an event with no bundles, $a(n')$. We have for each $Q[x := n]$, $\{\!|Q[x := n]|\!\}_{\mathcal{N}} = \langle \mathcal{E}_n, \mathsf{Init}_n, k_n \rangle$ and $\mathcal{E}'_n$ being the event structure generated by the causal semantics of $(\overline{\emptyset}, \underline{\emptyset}, \emptyset) \vdash Q[x := n]$ and by induction we have morphisms $f_n : P_B(\mathcal{E}_n) \to \mathcal{E}'_n$ and $f'_n :$

$\mathcal{E}'_n \to P_B(\mathcal{E}_n)$ such that $f_n \circ f'_n = 1_{P_B(\mathcal{E}_n)}$. This means we can define $f : P_B(\mathcal{E}) \to \mathcal{E}'$ as

$$f(e) = \begin{cases} \{(a(n), [P][Q[x := n]]), \emptyset)\} & \text{if } \lambda(e) = a(n) \text{ and no } e' \text{ exists such that } e' < e \\ \{(a(n), [P][Q[x := n]]), \emptyset)t \mid t \in f_n(e')\} & \text{if } e = (n, e') \text{ and there exists } e'' \text{ such that } e'' < e \end{cases}$$

This is clearly a morphism.
We also define $f' : \mathcal{E}' \to P_B(\mathcal{E})$ and

$$f'(e) = \begin{cases} e' & \text{if } ((a(n), [P][Q[x := n]]), \emptyset) \in e, \\ & \lambda(e') = a(n), \text{ and} \\ & \text{no } e'' \text{ exists such that } e'' < e' \\ f'_n(T) & \text{if } T = \{t \mid ((a(n), [P][Q[x := n]]), \emptyset)t \in e\} \neq \{\varepsilon\} \\ \bot & \text{otherwise} \end{cases}$$

This is clearly also a morphism and $f \circ f' = 1_{P_B(\mathcal{E})}$. Additionally, for any $e' \in E'$, if $f'(e')$ is defined, then $f(f'(e')) = e'$.

3. Suppose $P = \overline{a}(x).Q$. This case is similar to the previous.
4. Suppose $P = Q|R$. Then we have $\{\!|Q|\!\}_{\mathcal{N}} = \left\langle \mathcal{E}_Q, \mathsf{Init}_Q, k_Q \right\rangle$, $P_B(\mathcal{E}_Q) = \mathcal{E}_{P_Q}$, and $\mathcal{E}'_Q$ being the event structure generated by the causal semantics of $(\overline{\emptyset}, \underline{\emptyset}, \emptyset) \vdash Q$ as well as $\{\!|R|\!\}_{\mathcal{N}} = \langle \mathcal{E}_R, \mathsf{Init}_R, k_R \rangle$, $P_B(\mathcal{E}_R) = \mathcal{E}_{P_R}$, and $\mathcal{E}'_R$ being the event structure generated by the causal semantics of $(\overline{\emptyset}, \underline{\emptyset}, \emptyset) \vdash R$, and by induction there exist morphisms $f_Q : P_B(\mathcal{E}_Q) \to \mathcal{E}'_Q$, $f'_Q : \mathcal{E}'_Q \to P_B(\mathcal{E}_Q)$, $f_R : P_B(\mathcal{E}_R) \to \mathcal{E}'_R$, and $f'_R : \mathcal{E}'_R \to P_B(\mathcal{E}_R)$, such that $f_Q \circ f'_Q = 1_{P_B(\mathcal{E}_Q)}$ and $f_R \circ f'_R = 1_{P_B(\mathcal{E}_R)}$. We then define $f : P_B(\mathcal{E}) \to \mathcal{E}'$ as

$$f((X,e)) = \begin{cases} [(\mu, 0u, \emptyset)]_{\approx} & \text{if } X = \emptyset,\ e = (e_Q, *), \text{ and } f_Q(e_Q) = (\mu, u, \emptyset) \\ [(\mu, 1u, \emptyset)]_{\approx} & \text{if } X = \emptyset,\ e = (*, e_R), \text{ and } f_R(e_R) = (\mu, u, \emptyset) \\ [(\tau, \langle 0u_0, 1u_1 \rangle, \emptyset)]_{\approx} & \text{if } X = \emptyset,\ e = (e_Q, e_R),\ f_Q(e_Q) = (\alpha_0, u_0, \emptyset), \\ & \text{and } f_R(e_R) = (\alpha_1, u_1, \emptyset) \\ e' & \text{if } e = (e_Q, *),\ f_Q(e_Q) = (\mu, u, D), \\ & \mathsf{last}(e') = (\mu, 0u, D'), \\ & \text{and } e'' < e' \text{ iff} \\ & \exists (X''', e'''). f((X''', e''') = e'' \\ & \text{and } X''' \cup \{e'''\} \subseteq X \\ e' & \text{if } e = (*, e_R),\ f_R(e_R) = (\mu, u, D), \\ & \mathsf{last}(e') = (\mu, 1u, D'), \\ & \text{and } e'' < e' \text{ iff} \\ & \exists (X''', e'''). f((X''', e''') = e'' \\ & \text{and } X''' \cup \{e'''\} \subseteq X \\ e' & \text{if } e = (e_Q, e_R),\ f_Q(e_Q) = (\alpha_0, u_0, D_0), \\ & f_R(e_R) = (\alpha_1, u_1, D_1), \\ & \mathsf{last}(e') = (\tau, \langle 0u_0, 1u_1 \rangle, D'), \\ & \text{and } e'' < e' \text{ iff} \\ & \exists (X''', e'''). f((X''', e''') = e'' \\ & \text{and } X''' \cup \{e'''\} \subseteq X \end{cases}$$

To define the inverse of f, $f' : \mathcal{E}' \to P_B(\mathcal{E})$, we first define two helper functions g_0 and g_1, which extract the part of a trace which took place in each half of the process:

$$g_Q(t) = \begin{cases} \epsilon & \text{if } t = \epsilon \\ g_Q(t')(\mu, u, D') & \text{if } t = t'(\mu, 0u, D), D' = \{(n, u) \mid (n, 0u) \in D\} \\ g_Q(t') & \text{if } t = t'(\mu, 1u, D) \\ g_Q(t')(\alpha_0, u_0, D') & \text{if } t = t'(\tau, \left\langle 0l_0[P_0][P_0'], 1l_1[P_1][P_1'] \right\rangle, D), \\ & \exists i \in \{0, 1\}. P_i \xrightarrow{a(b)} P_i',\ P_{1-i} \xrightarrow{\overline{a}(b)} P_{1-i}', \\ & \alpha_i = a(b), \text{ and } \alpha_{1-i} = \overline{a}(b), \text{ and} \\ & D' = \{(n, u) \mid n \in no(\alpha_0) \text{ and } t = t_0(\overline{a}(n), 0u, D_0)t_1\} \end{cases}$$

$$g_R(t) = \begin{cases} \epsilon & \text{if } t = \epsilon \\ g_R(t')(\mu, u, D') & \text{if } t = t'(\mu, 1u, D), D' = \{(n,u) \mid (n,1u) \in D\} \\ g_R(t') & \text{if } t = t'(\mu, 0u, D) \\ g_R(t')(\alpha_1, u_1, D') & \text{if } t = t'(\tau, \left\langle 0l_0[P_0][P_0'], 1l_1[P_1][P_1'] \right\rangle, D), \\ & \exists i \in \{0,1\}. P_i \xrightarrow{a(b)} P_i',\ P_{1-i} \xrightarrow{\overline{a}(b)} P_{1-i}',\ \alpha_i = a(b), \\ & \text{and } \alpha_{1-i} = \overline{a}(b), \text{ and} \\ & D' = \{(n,u) \mid n \in no(\alpha_1) \text{ and } t = t_0(\overline{a}(n), 1u, D_1)t_1\} \end{cases}$$

It is clear that if t is a trace in $Q|R$, then $g_Q(t)$ is a trace in Q and $G_R(t)$ is a trace in R. We then define f' on e' with minimal representative $t(\mu, u, \emptyset)$:

$$f'(e') = \begin{cases} (X, (e_Q, e_R)) & \text{where if } u = 0u_0 \text{ or } u = \langle 0u_0, 1u_1 \rangle \text{ then} \\ & (X_Q, e_Q) = f_Q'([g_Q(t(\mu,u,\emptyset))]_\approx) \neq \bot, \\ & \text{and otherwise } e_Q = * \\ & \text{and if } u = 1u_1 \text{ or } u = \langle 0u_0, 1u_1 \rangle \text{ then} \\ & (X_R, e_R) = f_R'([g_R(t(\mu,u,\emptyset))]_\approx) \neq \bot, \\ & \text{and otherwise } e_R = * \\ & \text{if } t \text{ is empty, then } X = \emptyset, \\ & \text{otherwise } X = \bigcup_{\exists e'' < e'. f''(e'') = (X', e)} X' \cup \{e\} \\ \bot & \text{if } f_Q''([g_Q(t(\mu,u,\emptyset))]_\approx) = \bot \text{ or} \\ & f_R''([g_R(t(\mu,u,\emptyset))]_\approx) = \bot \\ & \text{in the relevant cases} \end{cases}$$

We must first prove that if $(X, e) \in E_P$, then $f((X, e)) \in E'$. By Theorems 4.8, 6.7, and 6.8 and Lemma 8.6, this only requires proving that there exists a trace $e_0, e_1, \ldots, e_n, e$ from $\emptyset$ where $\{e_0, e_1, \ldots, e_n\} = X$. If $X = \emptyset$, then there exist no bundles $X' \mapsto e$, and this is obvious. If $X \neq \emptyset$, then for each $e' \in X$, there exists $X' \subseteq X$ such that X' is a possible cause of e', so we order X based on $e'' < e'$ if $e'' \in X'$, and any sequence obeying this order is a trace. We then show that $X \xrightarrow{e}$ in the CS of E, This is obvious from Definition 5.5.

We then prove that for any $e_P = (X, e) \in E_P$, $f'(f(e_P)) = e_P$. We do this by induction on the size of X.

If $X = \emptyset$, then this is obvious.

If $X \neq \emptyset$, and for all $(X', e') \in E_P$ with X' smaller than X, $f'(f((X', e'))) = (X', e')$ then:

If $e = (e_Q, *)$, then by induction, there exists an event $(X_Q, e_Q) \in E_{Q_P}$ such that $f((X_Q, e_Q)) = [g_Q(f(e_P))]_\approx$ and $f'([g_Q(f(e_P))]_\approx) = (X_Q, e_Q)$. Therefore,

$f'(f(e_P)) = (X'', e)$ for some X''. For any $e'_P < e_P$, we know $f'(f(e'_P)) = e'_P$, and therefore we get $X'' = X$.

We then prove that f' is a morphism, if $e' \in E'$ and $f'(e') \neq \bot$, then there exists $f'(e') \in E_P$, and $f(f'(e')) = e'$. We do this by induction on the length of the minimal representatives of e', showing that that $f'(e') = (X_P, e_P) \in E_P$ and $f'_n = f' \upharpoonright \{e \mid \text{all minimal representatives of } e \text{ are of length } n\}$ is a morphism, and $f(f'_n(e') = e'$.

If the length of the minimal representatives of e' is 1, then we have three cases:

If $t = (\mu, 0u, \emptyset)$ is a minimal representative of e' and $f'(e') \neq \bot$, then by Theorems 4.8, 6.7, and 6.8 and Lemma 8.6 there must exist an event $e_Q \in E_Q$, such that $f'_Q((\mu, u, \emptyset)) = (X, e_Q)$, which can be performed from the configuration $\emptyset$ and has no link causes from R, meaning $X = \emptyset$ and $(\emptyset, (e_Q, *)) \in E_P$. Additionally, by induction, $f_Q(f'_Q([(\mu, u, \emptyset)]_\approx)) = [(\mu, u, \emptyset)]_\approx$, and therefore $f(f'_n(e') = e'$.

If $t = (\mu, 0u, \emptyset)$ is a minimal representative of e' then the argument is similar.

If $t = (\tau, \langle 0u_0, 1u_1 \rangle, \emptyset)$ is a minimal representative of e', then by induction we have events

$$(X_Q, e_Q) = f'_Q([g_Q((\tau, \langle u_0, u_1 \rangle, \emptyset))]_\approx) \in E_{P_Q}$$

and

$$(X_R, e_R) = f'_R([g_R((\tau, \langle u_0, u_1 \rangle, D))]_\approx) \in E_{P_R}$$

and by Theorems 4.8, 6.7, and 6.8 and Lemma 8.6 neither e_Q or e_R have any causes, meaning $X_Q = X_R = \emptyset$ and $(\emptyset, (e_Q, e_R))$. Additionally, by induction, we get

$$f_Q(f'_Q([g_Q((\tau, \langle u_0, u_1 \rangle, \emptyset))]_\approx)) = [g_Q((\tau, \langle u_0, u_1 \rangle, \emptyset))]_\approx$$

and

$$f_R(f'_R([g_R((\tau, \langle u_0, u_1 \rangle, \emptyset))]_\approx)) = [g_Q((\tau, \langle u_0, u_1 \rangle, \emptyset))]_\approx$$

and therefore $f(f'_n(e') = e'$.

We now prove that $f' \upharpoonright \{e \mid \text{all minimal representatives of } e \text{ are of length } 0\} = f'_0$ is a morphism.

(a) for all $e \in E'$, if $f'_0(e) \neq \bot$ then $\{e' \mid e <_P f'_0(e')\} = \emptyset = \{f(e'') \mid e'' <' e'\}$.

(b) for all $e, e' \in E'$, if $f'_0(e) \neq \bot \neq f'_0(e')$ and $f'_0(e) \sharp_P f'_0(e')$ then e has a minimum representative $(\mu, u, \emptyset)$ and e' has a minimum representative $(\mu', u', \emptyset)$ and either (a) $f'_0(e) = (\emptyset, (e_Q, *))$ and $f'_0(e') = (\emptyset, (e'_Q, *))$, (b) $f'_0(e) = (\emptyset, (e_Q, *))$ and $f'_0(e') = (\emptyset, (e'_Q, e'_R))$, (3) $f'_0(e) = (\emptyset, (e_Q, e_R))$ and $f'_0(e') = (\emptyset, (e'_Q, e'_R))$, (4) $f'_0(e) = (\emptyset, (e_Q, e_R))$ and $f'_0(e') = (\emptyset, (e'_Q, *))$, (5) $f'_0(e) = (\emptyset, (*, e_R))$ and $f'_0(e') = (\emptyset, (*, e'_R))$, (6) $f'_0(e) = (\emptyset, (*, e_R))$ and $f'_0(e') = (\emptyset, (e'_Q, e'_R))$, or (7) $f'_0(e) = (\emptyset, (e_Q, e_R))$ and $f'_0(e') = (\emptyset, (*, e'_R))$. If $f'_0(e) = (\emptyset, (e_Q, *))$ and $f'_0(e') = (\emptyset, (e'_Q, *))$, then by induction $(\emptyset, e_Q) \sharp_{Q_P} (\emptyset, e_Q)$ and $u = 0u_Q$ and $u' = 0u'_Q$, meaning $[(\mu, u_Q, \emptyset)]_\approx \sharp'_Q [(\mu' u'_Q, \emptyset)]_\approx$, and therefore $e' \sharp e''$.

If $f'_0(e) = (\emptyset, (e_Q, *))$ and $f'_0(e') = (\emptyset, (e'_Q, e'_R))$, then by induction we have $(\emptyset, e_Q) \sharp_{Q_P} (\emptyset, e_Q)$, $u = 0u_Q$, $u' = \left\langle 0u'_Q, 1u'_R \right\rangle$, and $[(\mu, u_Q, \emptyset)]_\approx \sharp'_Q [g_Q((\tau, u', \emptyset)]_\approx$. Therefore, $e' \sharp' e''$.
The other cases are similar.

(c) for all $e' \in F'$, if $f'_0(e) \neq \bot$ then $\{e \mid e \prec_P \underline{f'_0(e)}\} = \emptyset = \{f'_0(e'') \mid e'' \prec' \underline{e'}\}$.

(d) for all $e \in E'$ and $e' \in F'$, if $f'_0(e) \neq \bot \neq f'_0(e')$ then $f'_0(e) \not\rightarrow_P \overline{f'_0(e')}$.

(e) for all $e, e' \in E'$, if $f'_0(e) = f'_0(e') \neq \bot$ and $e \neq e'$ then $(\overline{\mu, u, \emptyset})$ is a minimal representative of e, $(\mu', u', \emptyset)$ is a minimal representative of e', and for some $i \in \{R, Q\}$, $f'_i([g_Q((\mu, u, \emptyset))]_\approx) = f'_i([g_i((\mu', u', \emptyset))]_\approx)$ and $[g_i((\mu, u, \emptyset))]_\approx \neq [g_i((\mu', u', \emptyset))]_\approx$.
If $i = Q$ then $[g_Q((\mu, u, \emptyset))]_\approx \sharp'_Q [g_Q((\mu', u', \emptyset))]_\approx$, and therefore $e \sharp_0 e'$. If $i = R$, the argument is similar.

(f) $f'_0(F') \subseteq F_P = E_P$.

If the length of the minimal representatives of e' is $n > 1$, and for any $e'' \in E'$ with minimal representatives of length $n' < n$, $f'(e'') = (X''_P, e''_P) \in E_P$, and we have a morphism $f'_{n-1} = f' \upharpoonright \{e \mid \text{all minimal representatives of } e \text{ are of length } n-1\}$, then we first show that $f'(e') \in E_P$, for which we again have 3 cases for e':

If $\mathsf{last}(e') = (\mu, 0u, D)$ and t of length n is a minimal representative of e', then $g_Q(t)$ is a trace in Q, and therefore by induction, $f'_Q(t) = (X_Q, e_Q) \in E_{P_Q}$, meaning there exists $(X, (e_Q, *)) \in E_P$ for some X. We therefore need to prove that $X = \bigcup_{\exists e'' < e'.\, f'(e'') = (X', e)} X' \cup \{e\}$ is a possible cause of $(e_Q, *)$ in $\mathcal{E}$. If $e'' < e'$, then e'' must have a minimum representative t' of length $n' < n$ such that $t'' \overline{\sim} t'$ is a prefix of t. By induction, this means $f'(e'') \in E_P$. By Lemma 8.2, we only need to prove the following:

(a) X is a configuration, which it is because $\{e'' \mid e'' < e'\}$ is conflict free, and by induction $\{f'_{n-1}(e'') \mid e'' < e'\}$ is conflict free, making X conflict free.

(b) X is reachable, which it is because for each $e \in X$, there exists a possible cause of e, $X' \subset X$, and if we order the events of X by inclusion of these associated possible causes, we get a partial order in which the events can be performed, similarly to Lemma 8.2.

(c) X is finite, which it is because $\{e'' \mid e'' <' e'\}$ is finite, and each potential cause of e'' is finite.

(d) $X \overset{e}{\rightarrow}$ by Theorems 4.8, 6.7, and 6.8 and Lemma 8.6.

(e) If $X'' \subset X$ is a reachable configuration, then there exists $e'' <' e'$ such that $f'_{n-1}(e'') = (X'', e)$ and $e \notin X''$, and since X'' is reachable, and therefore must include all causes of its events, we have minimum representative of e'', t'', such that $t = t_0 t_1$ and $t_0 \overline{\sim} t''$. Since t is a minimal representative, there must exist an action, (μ, u, D) in t_1 such that $\neg(\mathsf{last}(t'') I (\mu, u, D))$. If $(\mu, u, D) \neq \mathsf{last}(t)$, then by induction, since X'' is reachable, there exists e''' such that $\mathsf{last}(e''') = (\mu, u, D)$ and $e'' < e''' < e'$ and $f'_{n-1}(e''') = (X''', e'''_P)$ for $e_P \in X \setminus X''$. If $(\mu, u, D) = \mathsf{last}(t)$, then by Theorems 4.8, 6.7, and 6.8 and Lemma 8.6, $X'' \overset{e}{\not\rightarrow}$.

In addition, we must show that $f(f'(e')) = e'$. To do this, we merely need to show that there does not exist $e \in E'$ such that $\mathsf{last}(e) = \mathsf{last}(e')$ and for all $e'' \in E'$, $e'' < e$ iff $e'' < e'$, but $e \neq e'$. This is clearly not possible.
If $\mathsf{last}(e') = (\mu, 1u, D)$ or $\mathsf{last}(e') = (\tau, \langle 0u_0, 1u_1 \rangle, D)$, then the arguments are similar.
We then prove that $f'_n = f' \upharpoonright \{e \mid \text{all minimal representatives of } e \text{ are of length } n\}$ is a morphism:

(a) for all $e' \in E'$, if $f'_n(e') = (X, e) \neq \bot$ then $\{e'' \mid e'' <_P f_n(e')\} = \{(X'', e'') \mid \{e''\} \cup X'' \subseteq X\}$. We must now show that there cannot exist $(X'', e'') \in E_P$ such that $\{e''\} \cup X'' \subseteq X$ and there does not exist $e''' \in E'$ such that $f'_n(e''') = (X'', e'')$. This follows from Theorems 4.8, 6.7, and 6.8 and Lemmas 8.2 and 8.6.

(b) for all $e, e' \in E'$, if $f_n(e) = (X, e_P) \neq \bot \neq f_n(e') = (X', e'_P)$ and $f_n(e) \sharp_P f_n(e')$ then we have minimal representatives of e and e', t and t' and either $e_P \sharp e'_P$, or $e_P = e'_P$ and $X \neq X'$. In the first case, the conflict comes from either $f'_Q([g_Q(t)]_\approx) \sharp_{Q_P} f'_Q([g_Q(t')]_\approx)$ or $f'_R([g_R(t)]_\approx) \sharp_{R_P} f'_R([g_R(t')]_\approx)$, and by induction, either $[g_Q(t)]_\approx \sharp'_Q [g_Q(t')]_\approx$ or $[g_R(t)]_\approx \sharp'_Q [g_R(t')]_\approx$, meaning $e \sharp' e'$. In the second case, there must exist an action (μ, u, D) in t, which is not in t', and (μ', u', D') in t', which is not in t, and $\neg((\mu, u, D) I \mathsf{last}(t'))$ and $\neg((\mu', u', D') I \mathsf{last}(t))$. Therefore, $e \sharp' e'$.

(c) for all $e' \in F'$, if $f'_n(e') \neq \bot$ then $\{e \mid e \prec_P \underline{f'_n(e')}\} = \emptyset = \{f'_n(e) \mid e \prec' \underline{e'}\}$;

(d) for all $e \in E$ and $e' \in F_0$, if $f'_n(e) \neq \bot \neq f'_n(e')$ and $f'_n(e) \rhd_P \underline{f'_n(e')}$ then $f'_n(e') <_P f'_n(e)$, meaning as previously shown, $e' <' e$, and therefore $e \rhd' \underline{e'}$.

(e) for all $e, e' \in E'$, if $f'_n(e) = f'_n(e') \neq \bot$ then $e = e'$.

(f) $f(F_0) \subseteq F_1$.

We then prove that f is a morphism.

(a) for all $e \in E_P$, if $f(e) \neq \bot$ then $\{e' \mid e' <' f(e)\} = \{e' \mid \exists t' \in e'. \forall t \in f(e). \exists t'' \approx t. t'' \text{ is a prefix of } t'\} = \{f((X', e_P) \mid X' \cup \{e_P\} \subseteq X\} = \{f(e_P) \mid e_P <_P e\}$;

(b) for all $e_0, e_1 \in E_P$, if $f(e_0) \neq \bot \neq f(e_1)$ and $f(e_0) \sharp' f(e_1)$ then there exist events $e'_0 \leq' f(e_0)$ and $e'_1 \leq' f(e_1)$ and traces $t(\mu_0, u_0, D_0) \in e'_0$ and $t(\mu_1, u_1, D_1) \in e'_1$, such that $(\mu_0, u_0, D_0) \neq (\mu_1, u_1, D_1)$ and we do not have independence $(\mu_0, u_0, D_0) I (\mu_1, u_1, D_1)$. We therefore have $e''_0, e''_1 \in E_P$, such that $e''_0 \leq e_0$ and $f(e''_0) = e'_0$ and $e''_1 \leq e_1$ and $f(e''_1) = e'_1$. Since $\sharp_P$ is closed under conflict heredity, we only need to prove that $e''_0 \sharp e''_1$. If there exists $i \in 0, 1$ such that $u_0 = iu'_0$ and $u_1 = iu'_1$, then this follows from induction. Otherwise, there exists $i \in \{0, 1\}$ and $n \in \mathbb{N}$ such that $(n, u_i) \in D_{1-i}$, but this would require some (μ'_i, u_i, D'_i) to appear in t, which contradicts $t(\mu_i, u_i, D_i)$ being a trace.

(c) for all $e \in F_P$, if $f(e) \neq \bot$ then $\{e' \mid e' \prec' \underline{f(e)}\} = \emptyset = \{f(e') \mid e' \prec_P \underline{e}\}$;

(d) for all $e \in E_P$ and $e' \in F_P$, if $f(e) \neq \bot \neq \underline{f(e')}$ and $f(e) \rhd' \underline{f(e')}$ then $f(e') <' f(e)$, and from Item 1 we get $e' <_P e$, and therefore $e \rhd_P \underline{e'}$;

(e) for all $e, e' \in E_0$, if $f(e) = f(e') \neq \bot$ then $e = e'$;
(f) Straightforward
(g) Straightforward

5. Suppose $P = Q + R$. Then we have $\{\!|Q|\!\}_{\mathcal{N}} = \left\langle \mathcal{E}_Q, \mathsf{Init}_Q, k_Q \right\rangle$ and $\mathcal{E}'_Q$ being the event structure generated by the causal semantics of $(\overline{\emptyset}, \underline{\emptyset}, \emptyset) \vdash Q$ as well as $\{\!|R|\!\}_{\mathcal{N}} = \langle \mathcal{E}_R, \mathsf{Init}_R, k_R \rangle$ and $\mathcal{E}'_R$ being the event structure generated by the causal semantics of $(\overline{\emptyset}, \underline{\emptyset}, \emptyset) \vdash R$, and there exist morphisms $f_Q : P_B(\mathcal{E}_Q) \to \mathcal{E}'_Q$, $f'_Q : \mathcal{E}'_Q \to P_B(\mathcal{E}_Q)$, $f_R : P_B(\mathcal{E}_R) \to \mathcal{E}'_R$, and $f'_R : \mathcal{E}'_R \to P_B(\mathcal{E}_R)$ such that $f_Q \circ f'_Q = 1_{P_B(\mathcal{E}_Q)}$ and $f_R \circ f'_R = 1_{P_B(\mathcal{E}_R)}$. We define $f : P_B(\mathcal{E}) \to \mathcal{E}'$ as

$$f(e) = \begin{cases} \{(\alpha, [Q' + R][Q''], D)t \mid (\alpha, [Q'][Q''], D)t \in f_Q(e')\} & \text{if } e = (Q, e') \\ \{(\alpha, [Q + R'][R''], D)t \mid (\alpha, [R'][R''], D)t \in f_R(e')\} & \text{if } e = (R, e') \end{cases}$$

This is clearly a morphism.
We then define $f' : \mathcal{E}' \to P_B(\mathcal{E})$ as

$$f'(e) = \begin{cases} f'_Q(\{(\alpha, [Q'][Q''], D)t \mid (\alpha, [Q' + R][Q''], D)t \in e\}) & \\ \qquad \text{if } \exists \alpha, Q', Q'', D, t. & \\ \qquad (\alpha, [Q' + R][Q''], D)t \in e & \\ \qquad \text{and } (\alpha, [Q'][Q''], D)t \in e_Q \in E'_Q & \\ f'_R(\{(\alpha, [R'][R''], D)t \mid (\alpha, [Q + R'][R''], D)t \in e\}) & \\ \qquad \text{if } \exists \alpha, R', R'', D, t. & \\ \qquad (\alpha, [Q + R'][R''], D)t \in e & \\ \qquad \text{and } (\alpha, [R'][R''], D)t \in e_R \in E'_R & \\ \bot \qquad \text{otherwise} & \end{cases}$$

This is clearly also a morphism and since all names n P are distinct and therefore, $Q \neq R$, $f \circ f' = 1_{P_B(\mathcal{E})}$. Additionally, for any $e' \in E'$, if $f'(e')$ is defined, then $f(f'(e')) = e'$

6. Suppose $P = (\nu x)Q$. Isomorphism is preserved by restriction.

If P contains a nested replication of level n, and the result holds for all processes with a maximum level nested replication of $m < n$, then we again look at the structure of P. If $P \neq !P'$, then we can use the same arguments as above.

If $P = !P'$, then $\{\!|!_{\mathbf{x}} P'|\!\}_{\mathcal{N}} = \sup_{l \in \mathbb{N}} \{\!|!_{\mathbf{x}} P|\!\}_{(\mathcal{N}, l)} =$

$$\left\{\!\!\left| \ldots ((P'\{^{x_1,\ldots,x_k}/_{a_1,\ldots,a_k}\} \mid P'\{^{x_{k+1},\ldots,x_{2k}}/_{a_1,\ldots,a_k}\}) \mid P'\{^{x_{2k+1},\ldots,x_{3k}}/_{a_1,\ldots,a_k}\}) \ldots \right|\!\!\right\}_{\mathcal{N}}$$

where $\{a_1, \ldots, a_k\} = \mathsf{bn}(P')$, $x_1, \cdots \in \mathbf{x}$, and $x_i \neq x_j$ for $i \neq j$. The set of traces of $!P$ is the set of traces of $\ldots ((P' \mid P') \mid P') \mid \ldots$. By induction on the level of nesting replication, we get that the result holds for

$$\ldots ((P'\{^{x_1,\ldots,x_k}/_{a_1,\ldots,a_k}\} \mid P'\{^{x_{k+1},\ldots,x_{2k}}/_{a_1,\ldots,a_k}\}) \mid P'\{^{x_{2k+1},\ldots,x_{3k}}/_{a_1,\ldots,a_k}\}) \ldots$$

and we refer to the morphisms in question as f_x and f'_x. Since we have

$$\ldots ((P' \mid P') \mid P') \mid \ldots \equiv_\alpha$$

$$\ldots((P'\{^{x_1,\ldots,x_k}/_{a_1,\ldots,a_k}\}|P'\{^{x_{k+1},\ldots,x_{2k}}/_{a_1,\ldots,a_k}\})|P'\{^{x_{2k+1},\ldots,x_{3k}}/_{a_1,\ldots,a_k}\})\ldots$$

we know the traces of

$$\ldots((P'\{^{x_1,\ldots,x_k}/_{a_1,\ldots,a_k}\}|P'\{^{x_{k+1},\ldots,x_{2k}}/_{a_1,\ldots,a_k}\})|P'\{^{x_{2k+1},\ldots,x_{3k}}/_{a_1,\ldots,a_k}\})\ldots$$

are the same as the traces of $\ldots((P'|P')|P')|\ldots$. The result therefore follows from induction.

G.5. Proof of Lemma 8.10

PROOF. We obviously have a transition

$$\mathbf{H}_i \vdash P_i \xrightarrow[l_i[Q_i][Q'_i],D'_i]{\mu'_i} \mathbf{H}_{i+1} - (\mu_i, l_i[Q_i][Q'_i]) + (\mu'_i, l_i[Q_i][Q'_i[x := a]]) \vdash P'_{i+1}$$

created by simply receiving a different name.

We can then show that there exists a transition

$$\mathbf{H}_{i+1} - (\mu_i, l_i[Q_i][Q'_i]) + (\mu'_i, l_i[Q_i][Q'_i[x := a]]) \vdash P'_{i+1} \xrightarrow[u'_{i+1},D'_{i+1}]{\mu_i+1} \ldots \xrightarrow[u'_n,D'_n]{\mu'_n}$$

by simple induction on the length of the trace.

G.6. Proof of Theorem 8.11

PROOF. We say that we have $P_B(\mathcal{E}) = (E, F, <, \sharp, \prec, \rhd, \lambda, \mathsf{Act})$ and $\mathcal{E}' = (E', F', <', \sharp'$ $, \prec', \rhd', \lambda', \mathsf{Act}')$ and again prove this by induction on the structure and level of nesting of replication in P.

1. Suppose $P = 0$. Then both $\mathcal{E}$ and $\mathcal{E}'$ have no events.
2. Suppose $P = a(x).Q$. Then for each name n, $\mathcal{E}'$ has an event containing the trace $(a(n), [P][Q[x := n]]), \emptyset)$ and for each name $n' \in \mathcal{N}$, $\mathcal{E}$ has an event with no bundles, $a(n')$. We have for each $Q[x := n]$, $\{\!|Q[x := n]|\!\}_{\mathcal{N}} = \langle \mathcal{E}_n, \mathsf{Init}_n, k_n \rangle$, and $\mathcal{E}'_n$ being the event structure generated by the causal semantics of $(\overline{\emptyset}, \underline{\emptyset}, \emptyset) \vdash Q[x := n]$ and by induction we have morphisms $f_n : P_B(\mathcal{E}_n) \to \mathcal{E}'_n$ and $f'_n : \mathcal{E}'_n \to P_B(\mathcal{E}_n)$, defined as previously and $f''_n : \mathsf{Ir}(\mathcal{E}'_n) \to \mathsf{Ir}(\mathcal{E}_n)$ such that $f''_n \upharpoonright \mathsf{dom}(f'_n) = f'_n$.
   We then define f'' as

$$f''(e) = \begin{cases} f'_n(\{t \mid (a(n), u, D)t \in e\}) & \text{if } (a(n), u, D)t \in e \text{ for } t \neq \epsilon \\ & \text{and } n \in \mathcal{N} \\ e' & \text{if } (a(n), u, \emptyset) \in e,\ n \in \mathcal{N}, \\ & \lambda(e') = a(n), \text{ and} \\ & \text{no } e'' \text{ exists such that } e'' < e' \\ f'_x(\{t[m := x]_{(u-\emptyset)} \mid (\mu, u, D)t \in e\}) & \text{if } (a(m), u, D)t \in e \text{ for } t \neq \epsilon \\ & \text{and } m \notin \mathcal{N} \\ e' & \text{if } (a(m), u, \emptyset) \in e,\ m \notin \mathcal{N} \\ & \lambda(e') = a(x), \text{ and} \\ & \text{no } e'' \text{ exists such that } e'' < e' \end{cases}$$

Clearly $f'' \upharpoonright \mathsf{dom}(f') = f'$, and f'' is a morphism.

3. Suppose $P = \overline{a}(b).Q$. This case is similar to the previous.
4. Suppose $P = Q|R$. Then we have $\{\!|Q|\!\}_{\mathcal{N}} = \left\langle \mathcal{E}_Q, \mathsf{Init}_Q, k_Q \right\rangle$, and $\mathcal{E}'_Q$ being the event structure generated by the causal semantics of $(\overline{\emptyset}, \underline{\emptyset}, \emptyset) \vdash Q$ as well as $\{\!|R|\!\}_{\mathcal{N}} = \langle \mathcal{E}_R, \mathsf{Init}_R, k_R \rangle$, and $\mathcal{E}'_R$ being the event structure generated by the causal semantics of $(\overline{\emptyset}, \underline{\emptyset}, \emptyset) \vdash R$, and by induction we have the morphisms $f_Q : P_B(\mathcal{E}_Q) \to \mathcal{E}'_Q$ and $f'_Q : \mathcal{E}'_Q \to P_B(\mathcal{E}_Q)$ and $f''_Q : \mathsf{Ir}(\mathcal{E}'_Q) \to \mathsf{Ir}(\mathcal{E}_Q)$ and $f_R : P_B(\mathcal{E}_R) \to \mathcal{E}'_R$, $f'_R : \mathcal{E}'_R \to P_B(\mathcal{E}_R)$ and $f''_R : \mathsf{Ir}(\mathcal{E}'_R) \to \mathsf{Ir}(\mathcal{E}_R)$, with f_Q, f'_Q, f_R, and f'_R as before, and $f''_Q \upharpoonright \mathsf{dom}(f'_Q) = f'_Q$ and $f''_R \upharpoonright \mathsf{dom}(f'_R) = f'_R$. To define f'', we again use the two helper functions g_0 and g_1 defined in the parallel case of the previous proof, which extract the part of a trace which took place in each half of the process:

$$g_Q(t) = \begin{cases} \epsilon & \text{if } t = \epsilon \\ g_Q(t')(\mu, u, D') & \text{if } t = t'(\mu, 0u, D), \text{ and } D' = \{(n,u) \mid (n, 0u) \in D\} \\ g_Q(t') & \text{if } t = t'(\mu, 1u, D) \\ g_Q(t')(\alpha_0, u_0, D') & \text{if } t = t'(\tau, \left\langle 0l_0[P_0][P'_0], 1l_1[P_1][P'_1] \right\rangle, D), \\ & \exists i \in \{0,1\}. P_i \xrightarrow{a(b)} P'_i,\ P_{1-i} \xrightarrow{\overline{a}(b)} P'_{1-i},\ \alpha_i = a(b), \\ & \text{and } \alpha_{1-i} = \overline{a}(b), \text{ and} \\ & D' = \{(n,u) \mid n \in no(\alpha_0) \text{ and } t = t_0(\overline{a}(n), 0u, D_0)t_1\} \end{cases}$$

$$g_R(t) = \begin{cases} \epsilon & \text{if } t = \epsilon \\ g_R(t')(\mu, u, D') & \text{if } t = t'(\mu, 1u, D), \text{ and } D' = \{(n,u) \mid (n, 1u) \in D\} \\ g_R(t') & \text{if } t = t'(\mu, 0u, D) \\ g_R(t')(\alpha_1, u_1, D') & \text{if } t = t'(\tau, \left\langle 0l_0[P_0][P'_0], 1l_1[P_1][P'_1] \right\rangle, D), \\ & \exists i \in \{0,1\}. P_i \xrightarrow{a(b)} P'_i,\ P_{1-i} \xrightarrow{\overline{a}(b)} P'_{1-i},\ \alpha_i = a(b), \\ & \text{and } \alpha_{1-i} = \overline{a}(b), \text{ and} \\ & D' = \{(n,u) \mid n \in no(\alpha_1) \text{ and } t = t_0(\overline{a}(n), 1u, D_1)t_1\} \end{cases}$$

It is clear that if t is a trace in $Q|R$, then $g_Q(t)$ is a trace in Q and $G_R(t)$ is a trace

in R. We define f'' of an event e' with a minimal trace $t(\mu, u, \emptyset)$:

$$f''(e') = \begin{cases} (X, (e_Q, e_R)) & \text{where if } u = 0u_0 \text{ or } u = \langle 0u_0, 1u_1\rangle \text{ then} \\ & (X_Q, e_Q) = f''_Q([g_Q(t(\mu,u,\emptyset))]_\approx) \neq \bot, \\ & \text{and otherwise } e_Q = * \\ & \text{and if } u = 1u_1 \text{ or } u = \langle 0u_0, 1u_1\rangle \text{ then} \\ & (X_R, e_R) = f''_R([g_R(t(\mu,u,\emptyset))]_\approx) \neq \bot, \\ & \text{and otherwise } e_R = * \\ & \text{if } t \text{ is empty, then } X = \emptyset, \\ & \text{otherwise } X = \bigcup_{\exists e'' < e'.\, f''(e'') = (X', e)} X' \cup \{e\} \\ \bot & \text{if } f''_Q([g_Q(t(\mu,u,\emptyset))]_\approx) = \bot \text{ or} \\ & f''_R([g_R(t(\mu,u,\emptyset))]_\approx) = \bot \\ & \text{in the relevant cases} \end{cases}$$

Clearly, $f'' \restriction \mathsf{dom}(f') = f'$, and the proof of f'' being a morphism is similar to the proof of f' being a morphism in the previous theorem.

5. Suppose $P = Q + R$. Then we have $\{\!|Q|\!\}_{\mathcal{N}} = \left\langle \mathcal{E}_Q, \mathsf{Init}_Q, k_Q \right\rangle$, and $\mathcal{E}'_Q$ being the event structure generated by the causal semantics of $(\overline{\emptyset}, \underline{\emptyset}, \emptyset) \vdash Q$ as well as $\{\!|R|\!\}_{\mathcal{N}} = \langle \mathcal{E}_R, \mathsf{Init}_R, k_R \rangle$, and $\mathcal{E}'_R$ being the event structure generated by the causal semantics of $(\overline{\emptyset}, \underline{\emptyset}, \emptyset) \vdash R$, and by inductions we have morphisms $f_Q : P_B(\mathcal{E}_Q) \to \mathcal{E}'_Q$, $f'_Q : \mathcal{E}'_Q \to P_B(\mathcal{E}_Q)$, $f''_Q : \mathsf{Ir}(\mathcal{E}'_Q) \to \mathsf{Ir}(\mathcal{E}_Q)$, $f_R : P_B(\mathcal{E}_R) \to \mathcal{E}'_R$, $f'_R : \mathcal{E}'_R \to P_B(\mathcal{E}_R)$, $f''_R : \mathsf{Ir}(\mathcal{E}'_R) \to \mathsf{Ir}(\mathcal{E}_R)$, with f_Q, f'_Q, f_R, and f'_R as before, and $f''_Q \restriction \mathsf{dom}(f'_Q) = f'_Q$ and $f''_R \restriction \mathsf{dom}(f'_R) = f'_R$. We then define

$$f''(e) = \begin{cases} f''_Q(\{(\alpha, [Q'][Q''], D)t \mid (\alpha, [Q' + R][Q''], D)t \in e\}) \\ \qquad \text{if } \exists \alpha, Q', Q'', D, t. \\ \qquad (\alpha, [Q' + R][Q''], D)t \in e \\ \qquad \text{and } (\alpha, [Q'][Q''], D)t \in e_Q \in E'_Q \\ f''_R(\{(\alpha, [R'][R''], D)t \mid (\alpha, [Q + R'][R''], D)t \in e\}) \\ \qquad \text{if } \exists \alpha, R', R'', D, t. \\ \qquad (\alpha, [Q + R'][R''], D)t \in e \\ \qquad \text{and } (\alpha, [R'][R''], D)t \in e_R \in E'_R \end{cases}$$

This obviously is a morphism and $f'' \restriction \mathsf{dom}(f') = f'$.

6. Suppose $P = (\nu x)Q$. Then Then we have $\{\!|Q|\!\}_{\mathcal{N}} = \left\langle \mathcal{E}_Q, \mathsf{Init}_Q, k_Q \right\rangle$, and $\mathcal{E}'_Q$ being the event structure generated by the causal semantics of $(\overline{\emptyset}, \underline{\emptyset}, \emptyset) \vdash Q$ and by inductions we have morphisms $f_Q : P_B(\mathcal{E}_Q) \to \mathcal{E}'_Q$, $f'_Q : \mathcal{E}'_Q \to P_B(\mathcal{E}_Q)$, and $f''_Q : \mathsf{Ir}(\mathcal{E}'_Q) \to \mathsf{Ir}(\mathcal{E}_Q)$, with f_Q and f'_Q as before, and $f''_Q \restriction \mathsf{dom}(f'_Q) = f'_Q$. The traces of P will be a subset of the traces of Q, and we define f'' as:

$$f''(e) = f''_Q(e') \text{ if } e' \subseteq e \text{ and } e' \in E'_Q$$

7. Suppose $P = !Q$. Then the argument is much the same as in Theorem 8.7.