




Diamonds for Security: A Non-Interleaving Operational Semantics for the Applied Pi-Calculus

Clément Aubert   

Augusta University, Augusta, GA, USA

Ross Horne  

University of Luxembourg, Luxembourg

Christian Johansen   

NTNU – Norwegian University of Science and Technology, Gjøvik, Norway

Abstract

We introduce a non-interleaving structural operational semantics for the applied π -calculus and prove that it satisfies the properties expected of a labelled asynchronous transition system (LATS). LATS have well-studied relations with other standard non-interleaving models, such as Mazurkiewicz traces or event structures, and are a natural extension of labelled transition systems where the independence of transitions is made explicit. We build on a considerable body of literature on located semantics for process algebras and adopt a static view on locations to identify the parallel processes that perform a transition. By lifting, in this way, work on CCS and π -calculus to the applied π -calculus, we lay down a principled foundation for reusing verification techniques such as partial-order reduction and non-interleaving equivalences in the field of security. The key technical device we develop is the notion of located aliases to refer unambiguously to a specific output originating from a specific process. This light mechanism ensures stability, avoiding disjunctive causality problems that parallel extrusion incurs in similar non-interleaving semantics for the π -calculus.

2012 ACM Subject Classification Theory of computation \rightarrow Program semantics; Theory of computation \rightarrow Concurrency; Theory of computation \rightarrow Process calculi

Keywords and phrases Security, Processes, Structural operational semantics, Asynchronous transition systems, Applied pi-calculus

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2022.30

Acknowledgements The authors wish to express their gratitude to the reviewers for their recommendations, that helped us improve our presentation.

1 Introduction

The purpose of this paper is to give a principled foundation for methods from concurrency theory that are gaining traction in the verification of security properties. Tools used in the security domain have applied partial-order-based techniques, notably partial-order reduction (POR), to improve their efficiency, thereby increasing the size of the systems that can be analyzed [6, 7, 8, 17, 18, 21, 27, 38]. To date, much of that work is tool-driven and guided by examples. However, without a solid foundation, we cannot be certain that the methods employed are correct, and hence, if an untested example is presented, the tool might produce incorrect results or fail to apply reductions where they might well have been applied.

This paper fulfils what we believe to be an important role: drawing from decades of theory on the topic of non-interleaving semantics for process algebras, we bring essential concurrency concepts to the security verification community. In particular, we aim for a semantics that is operational, rather than denotational, to stay close to the existing semantics employed by the main tools in the security community, so that our work may easily be adopted. As in the non-interleaving tradition, we define a concept of *event* as an enhancement of actions. The



© Clément Aubert, Ross Horne, and Christian Johansen;
licensed under Creative Commons License CC-BY 4.0

33rd International Conference on Concurrency Theory (CONCUR 2022).

Editors: Bartek Klin, Sławomir Lasota, and Anca Muscholl; Article No. 30; pp. 30:1–30:26



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

notion of independence is then defined on events in a straightforward way using the structure of the events only. This enables the implementation of techniques such as partial-order reduction on the fly.

The applied π -calculus [1, 43] has become instrumental in leveraging theoretical process calculi to certify and verify security protocols [4, 10, 14, 31, 33, 36]. Its syntax can describe a variety of interactions between the processes involved in a security protocol, stressing the study of the communication of complex messages. Abstracting away from cryptographic primitives with an equational theory allows tools to focus on problems arising due to the information flow in security protocols, while its flexibility allows a large variety of situations to be modelled. Its inductive structure lends itself to automation of the analysis and verification of protocols used in production thanks to tools such as ProVerif [10], DEEPSEC [17], Akiss [16], Sapic [35], SAT-Equiv [19] and SPEC [46], which have flavours of the applied π -calculus as their input language.

Labelled asynchronous transition systems (LATS) are a non-interleaving model of concurrency, which, because they extend transition systems in a natural way, are suited to being the objects generated by our structural operational semantics for the applied π -calculus. The word *asynchronous* does not refer to the type of communication between processes: it is a nod to the Petri net literature, where independent transitions are said to act asynchronously [41]. LATS were introduced by Bednarczyk [9] and Shields [45], and have been well studied in the concurrency literature [28, 29, 51]. They are generally required to satisfy properties such as *event determinism*, and to provide an independence relation that satisfies what we call *concurrency diamonds*. Developing a LATS is also an important prerequisite for further methods to be developed such as non-interleaving process equivalences.

Endowing the applied π -calculus with a structural operational semantics that defines a labelled asynchronous transition system gives us a concrete path towards applying non-interleaving concurrency techniques to security problems such as the verification of security protocols. This requires us to import and mix ideas from different communities and lines of work, and to carefully design “the right fit” to prove seamlessly the desired properties of a LATS. We believe our proposal to be not only elegant, but also enlightening in the way it sidesteps “traditional” problems stemming from the π -calculus, such as the need to represent some forms of extrusion with disjunctive causality [20, 30]. In particular, in our theory there is never any ambiguity about which outputs an event is causally dependent on, in contrast to the traditional semantics for the π -calculus [11] where, if two outputs concurrently extrude the same name that is then later used, then the semantics does not record from which output the name originates.

Our proposal is also lightweight – our *events* consist simply of a location (a binary string indicating where in the binary tree of parallel and choice components the event originates) and an action label – and provides event determinism almost for free. Avoiding disjunctive causality will allow our model to be related to more standard concurrency models such as safe Petri nets or prime event structures.

Outline. Sect. 2 recalls the abstract concept of a LATS. Sect. 3 introduces a syntax for the applied π -calculus. Sect. 4 explains the design of our non-interleaving structural operational semantics. Sect. 5 presents the main result: that our non-interleaving structural operational semantics is a labelled asynchronous transition system. Sect. 6 situates this work in the literature, particularly in relation to POR for the applied π -calculus. The appendices provide example derivations and outline the key elements of the proof of the main theorem.

2 Our Target Model: Labelled Asynchronous Transition Systems

A labelled asynchronous transition system is a labelled transition system enriched with an independence relation on events I that satisfies what we call *diamond* properties. Standard labelled transition systems, such as those obtained from structural operational semantics of process algebras such as CCS, π -calculus, and applied π -calculus, are labelled with *actions*, indicating, e.g., the input or output action of a process. Labelled *asynchronous* transition systems (LATS), in contrast, are labelled with *events*, which contain more information, sufficient to distinguish between events that are triggered in different ways but share the same action label. We denote states (which usually in process algebras are process terms) by capital letters, events by small letters, and transitions between states labelled with an event as $A \xrightarrow{e} B$. We illustrate the properties of LATS using standard π -calculus notation. The reader familiar with the π -calculus may safely skip the following info box.

Key features of the π -calculus: We assume for now only some familiarity with π -calculus features, some of which we informally recall here. Later, in Sect. 3, we extend this to the applied π -calculus and define formally its semantics:

- Name restriction $\nu x.P$ binds occurrences of variable x in P , indicating that x is a freshly generated name. Importantly, an observer (who may be an attacker) cannot know (e.g., by guessing) which name was generated, without intercepting a communication of this name on a public channel.
 - An output prefix $\bar{x}(y).P$ indicates that the variable y is output on a channel x , before continuing to execute process P . Such outputs may only be observed if the channel is known to the observer, either because x is a free variable or is a fresh name that has previously been output.
 - An input prefix $x(y).P$ indicates that a channel x is used to receive a message, which is always a variable representing a name in the π -calculus. Notably, a fresh name may only be received if it has previously been output. The process then continues to execute P , but with occurrences of variable y in P replaced by the variable received, e.g., as $P\{z/y\}$ if z was received along x .
 - Parallel composition $P \mid Q$ indicates that processes P and Q execute in separate locations, possibly communicating with each other by synchronising an input and output on the same channel name, whether or not the channel is known to an observer. The precise meaning of parallel composition is the main subject of the interleaving/non-interleaving spectrum of process semantics.
 - The match prefix $[x = y]P$ should be read as, “if x and y are the same variable (due to a prior input action for example) then P can execute.”
 - The process 0 represents termination. We follow standard conventions such as omitting the deadlock process 0 when it is preceded by an action (e.g., output or input) prefix.
- A distinguishing feature of the π -calculus is that the scope of a name restriction is mobile in the sense that it can expand when a message containing the variable it binds is output. We assume that action prefixes and name restriction bind stronger than parallel composition.

To begin with, a LATS must satisfy the following property, which ensures that, in any given state, any two (co-initial) transitions labelled with the same event are actually the same transition (up to some minimal congruence relation \equiv , quotienting the set of states).

► **Definition 1** (event determinism). *A LATS satisfies event determinism whenever*

$$\text{If } A_0 \equiv A_1, A_0 \xrightarrow{e} B_0 \text{ and } A_1 \xrightarrow{e} B_1, \text{ then } B_0 \equiv B_1.$$

For example, for a processes $\bar{c}\langle n \rangle \mid \bar{c}\langle n \rangle$ consisting of two parallel threads both sending n on channel c , a LATS must label the two output transitions differently, which is not guaranteed by standard labelled transition systems.

The other two properties a LATS must satisfy are the diamond properties (sometimes called “sideways diamond” [42] and “square property” [37]), which ensure that events considered to be independent with respect to the independence relation I can permute. The first diamond property ensures that two independent events labelling (co-initial) transitions from the same state can be performed in either order without affecting the outcomes.

► **Definition 2** (diamond property 1). *A LATS satisfies diamond property 1 whenever*

$$\text{If } e_0 I e_1, A \xrightarrow{e_0} B_0, \text{ and } A \xrightarrow{e_1} B_1 \text{ then } \exists C_0, C_1 \text{ s.t. } B_0 \xrightarrow{e_1} C_0, B_1 \xrightarrow{e_0} C_1 \text{ and } C_0 \equiv C_1.$$

The second diamond property concerns independent events labelling (composable) transitions in *consecutive* states, ensuring such transitions may be permuted.

► **Definition 3** (diamond property 2). *A LATS satisfies diamond property 2 whenever*

$$\text{If } e_0 I e_1, A \xrightarrow{e_0} B_0, \text{ and } B_0 \xrightarrow{e_1} C_0, \text{ then } \exists B_1, C_1 \text{ s.t. } A \xrightarrow{e_1} B_1, B_1 \xrightarrow{e_0} C_1 \text{ and } C_0 \equiv C_1.$$

Observe that all of the above properties only concern co-initial or composable transitions, i.e., transitions from the same state or adjacent states. We use the following example to illustrate why this is significant: $\nu n.(\bar{c}\langle n \rangle \mid \overline{c(x)}.[x = n]\overline{\text{ok}}\langle \text{ok} \rangle)$. For any execution of this process in which the event corresponding to $\overline{\text{ok}}\langle \text{ok} \rangle$ occurs, the two events corresponding to $\bar{c}\langle n \rangle$ and $c(x)$ must have both occurred previously. Since we work with an *early* semantics, strictly speaking we have one event for each possible instantiation of the variable x in the input $c(x)$. *Structural* (a.k.a. *prefixing*) causality ensures that the event corresponding to $\overline{\text{ok}}\langle \text{ok} \rangle$ should not be independent from any of the input events corresponding to $c(x)$, since this part (or *location*) of the process must execute the latter to be able to access the former. *Link* (a.k.a. *name*) causality ensures that the event corresponding to $\bar{c}\langle n \rangle$ should not be independent from the input event corresponding to $c(x)$ when x is instantiated with the name n , which would also enable the guard $x = n$. Taken together, those two causalities ensure that our expectations regarding the concurrency inherent in π -calculi are met.

In contrast to the above observations, although the events corresponding to $\bar{c}\langle n \rangle$ and $\overline{\text{ok}}\langle \text{ok} \rangle$, in the above example, are causally dependent on each other by transitivity, when defining a LATS, we are free to allow them to be defined as “independent”, since these events can never be executed concurrently or consecutively, and hence they will never be considered together in a diamond property. This is a reason why LATS are suited for defining a non-interleaving structural operational semantics, as there is no need to compute global dependencies between events: local calculations are enough to determine whether consecutive events are concurrent. If global dependencies are required, they are established by unfoldings of LATS into event structures and Petri nets [39].

The relatively light requirements on the independence relation, as explained above, justifies why LATS offer an attractive take on structural operational semantics. A structural operational semantics makes transitions easy to compute, and an easy to compute independence relation facilitates partial-order reduction, that drastically reduces the number of states to explore when verifying concurrent processes [3, 18]. An easily calculated independence relation also facilitates the checking of non-interleaving variants of equivalences, e.g., distinguishing $\bar{c}\langle n \rangle \mid \bar{c}\langle n \rangle$ from $\bar{c}\langle n \rangle.\bar{c}\langle n \rangle$ (the *autoconcurrency* problem). Causality in the π -calculus has been explored in related work [11, 20, 22, 34, 40].

are enlarged to include the substitution, so that they may bind variables in both the substitutions representing messages that have been output and in the continuation processes. In an extended process $A = \nu \vec{x}.(\sigma \mid P)$, we assume, in this work, a *normal form*, where aliases do not appear in processes. This has the effect that the active substitution σ in the extended process A has already been applied to P .

► **Definition 4** (free variables and aliases). *A variable x (resp. an alias α) is free in a message M if $x \in \text{fv}(M)$ (resp. $\alpha \in \text{fa}(M)$) for*

$$\begin{aligned} \text{fv}(f(M_1, \dots, M_n)) &= \cup_{i=1}^n \text{fv}(M_i) & \text{fv}(x) &= \{x\} & \text{fv}(\alpha) &= \emptyset \\ \text{fa}(f(M_1, \dots, M_n)) &= \cup_{i=1}^n \text{fa}(M_i) & \text{fa}(x) &= \emptyset & \text{fa}(\alpha) &= \{\alpha\}. \end{aligned}$$

The fv function extends in the standard way to (extended) processes, letting $\text{fv}(\nu x.P) = \text{fv}(P) \setminus \{x\}$ and $\text{fv}(M(x).P) = \text{fv}(M) \cup (\text{fv}(P) \setminus \{x\})$, and similarly for $\text{fv}(A)$.

► **Definition 5** (fresh). *We say a variable x is fresh for a message M (resp. process P , extended process A), written $x \# M$ (resp. $x \# P$, $x \# A$) whenever $x \notin \text{fv}(M)$ (resp. $x \notin \text{fv}(P)$, $x \notin \text{fv}(A)$), and similarly for aliases. Freshness extends point-wise to lists of entities, i.e., $x_1, x_2, \dots, x_m \# M_1, M_2, \dots, M_n$, denotes the conjunction of all $x_i \# M_j$ for all $1 \leq i \leq m$ and $1 \leq j \leq n$.*

► **Definition 6** (α -equivalence). *We define α -equivalence (denoted \equiv_α) for variables only (not aliases which are fixed “addresses”) as the least congruence (a reflexive, transitive, and symmetric relation preserved in all contexts) such that, whenever $z \# \nu x.P$, we have $\nu x.P \equiv_\alpha \nu z.(P\{z/x\})$ and $M(x).P \equiv_\alpha M(z).(P\{z/x\})$. Similarly, for extended processes, we have the least congruence such that, whenever $z \# \nu x.A$, we have $\nu x.A \equiv_\alpha \nu z.(A\{z/x\})$.*

► **Definition 7** (capture-avoiding substitutions). *Restriction is such that $\theta \upharpoonright_{\vec{\alpha}}(x) = \theta(x)$ if $x \in \vec{\alpha}$ and x otherwise. Capture-avoiding substitutions are defined for processes such that for any $z \# \text{dom}(\sigma), \text{ran}(\sigma), \nu x.P$, we have $(M(x).P)\sigma \equiv_\alpha M\sigma(z).P\{z/x\}\sigma$ and $(\nu x.P)\sigma \equiv_\alpha \nu z.P\{z/x\}\sigma$. For extended processes, it is defined such that $(\nu x.A)\rho \equiv_\alpha \nu z.(A(\{z/x\} \circ \rho))$ and $(\sigma \mid P)\rho = (\sigma \circ \rho \upharpoonright_{\text{dom}(\sigma)} \mid P\rho)$, for $z \# \text{dom}(\rho), \text{ran}(\rho), \nu x.A$.*

► **Definition 8** (structural congruence). *Our minimal structural congruence (denoted \equiv) is the least equivalence relation on extended processes extending α -equivalence such that whenever $\sigma = \theta$ (i.e., the substitutions denote the same function), $P \equiv_\alpha Q$ and $A \equiv B$, we have:*

$$\sigma \mid P \equiv \theta \mid Q \qquad \nu x.A \equiv \nu x.B \qquad \nu x.\nu z.A \equiv \nu z.\nu x.A$$

Notice that we did not include equations such as $P \mid Q \equiv Q \mid P$, $(P \mid Q) \mid R \equiv P \mid (Q \mid R)$ or $P \mid 0 \equiv P$, for reasons that will become clear in Sect. 4.2. Many similar systems (sometimes called *proved* transition systems) miss a structural congruence altogether [15, 24, 26], or miss the associativity and commutativity of the parallel composition [25, p. 242], since they can alter the label of the transition and complicate tracking the source of an action, yet can be recovered by a suitable observational equivalence (e.g., a non-interleaving bisimilarity).

4 The Design of a Non-interleaving Structural Operational Semantics

The located structural operational semantic rules introduced in Figs. 2 and 3 adapt the recent semantics developed for the applied π -calculus in [31, 32], which can be obtained from the one here by simply ignoring the information in red and any other information beneath the arrow in labelled transitions. To obtain a structural operational semantics for

$$\begin{array}{c}
\frac{M =_E K}{K(x).P \xrightarrow[\square]{MN} \text{id} \mid P\{N/x\}} \text{INP} \qquad \frac{M =_E K}{\overline{K}(N).P \xrightarrow[\square]{\overline{M}(\lambda)} \{N/\lambda\} \mid P} \text{OUT} \\
\\
\frac{P \xrightarrow[u]{\pi} \nu \vec{x}.(\sigma \mid R) \quad \vec{x} \# Q}{P \mid Q \xrightarrow[\mathbf{0}_u]{\pi} \nu \vec{x}.(\sigma \mid R \mid Q)} \text{PAR-L} \qquad \frac{Q \xrightarrow[u]{\pi} \nu \vec{x}.(\sigma \mid R) \quad \vec{x} \# P}{P \mid Q \xrightarrow[\mathbf{1}_u]{\pi} \nu \vec{x}.(\sigma \mid P \mid R)} \text{PAR-R} \\
\\
\frac{P \xrightarrow[u]{\pi} A \quad x \# \text{fv}(\pi)}{\nu x.P \xrightarrow[u]{\pi} \nu x.A} \text{EXTRUDE} \qquad \frac{A \xrightarrow[u]{\pi} B \quad x \# \text{fv}(\pi)}{\nu x.A \xrightarrow[u]{\pi} \nu x.B} \text{RES} \\
\\
\frac{P \xrightarrow[\mathbf{s}[s']]{\overline{M}\sigma(\lambda)} \nu \vec{x}.(\{N/\lambda\} \mid Q) \quad \vec{x} \# \text{ran}(\sigma) \quad \text{fa}(M) \subseteq \text{dom}(\sigma) \quad \mathbf{s}\lambda \# \text{dom}(\sigma)}{\sigma \mid P \xrightarrow[\mathbf{s}[s']]{\overline{M}(s\lambda)} \nu \vec{x}.(\sigma \circ \{N/\mathbf{s}\lambda\} \mid Q)} \text{ALIAS-OUT} \\
\\
\frac{P \xrightarrow[u]{\pi\sigma} \nu \vec{x}.(\text{id} \mid Q) \quad \vec{x} \# \text{ran}(\sigma) \quad \text{fa}(\pi) \subseteq \text{dom}(\sigma)}{\sigma \mid P \xrightarrow[u]{\pi} \nu \vec{x}.(\sigma \mid Q)} \text{ALIAS-FREE} \\
\\
\frac{G \xrightarrow[t]{\pi} A}{G + H \xrightarrow[\mathbf{0}t]{\pi} A} \text{SUM-L} \qquad \frac{H \xrightarrow[t]{\pi} A}{G + H \xrightarrow[\mathbf{1}t]{\pi} A} \text{SUM-R} \qquad \frac{P \mid !P \xrightarrow[u]{\pi} A}{!P \xrightarrow[u]{\pi} A} \text{BANG} \\
\\
\frac{P \xrightarrow[u]{\pi} A \quad M =_E N}{[M = N]P \xrightarrow[u]{\pi} A} \text{MAT} \qquad \frac{P \xrightarrow[u]{\pi} A \quad M \neq_E N}{[M \neq N]P \xrightarrow[u]{\pi} A} \text{MISMAT}
\end{array}$$

■ **Figure 2** An *early* located non-interleaving structural operational semantics.

the π -calculus in this style, simply assume that all messages M , N , etc., are variables, and that two variables are equal only if they are the same variable. After briefly introducing our structural operational semantics, we explain our design choices in the subsections that follow.

Action labels range over π . A *free input* action label MN indicates the input of message N on channel M . A *bound output* action label $\overline{M}(\alpha)$ indicates the output of something on channel M where the message we output is assigned the alias α , which can be used to refer to that message in the future by the observer, or a direct communication τ that the observer does not intercept. This is the reason why the substitution is applied to the continuation in the INP rule, but “stored” in the substitution in the OUT rule. The label τ denotes an internal communication, and is used in the synchronisation rules presented in Fig. 3.

The functions for free variables and free aliases extend to labels as follows.

$$\text{fv}(\pi) = \begin{cases} \text{fv}(M) \cup \text{fv}(N) & \text{if } \pi = MN \\ \text{fv}(M) & \text{if } \pi = \overline{M}(\alpha) \\ \emptyset & \text{if } \pi = \tau \end{cases} \qquad \text{fa}(\pi) = \begin{cases} \text{fa}(M) \cup \text{fa}(N) & \text{if } \pi = MN \\ \text{fa}(M) & \text{if } \pi = \overline{M}(\alpha) \\ \emptyset & \text{if } \pi = \tau \end{cases}$$

Readers familiar with labelled transition systems will immediately notice the additional annotation below transitions, that uses prefixes composed of **0s** and **1s**. This indicates the *location* of the parallel sub-process (a.k.a. component or thread) performing the action:

$$\begin{array}{c}
\frac{P \xrightarrow[\ell_0]{\overline{M}(\lambda)} \nu \vec{y}.(\{N/\lambda\} | P') \quad Q \xrightarrow[\ell_1]{MN} \nu \vec{w}.(\text{id} | Q') \quad \vec{y} \# Q \quad \vec{w} \# P, \vec{y}}{P | Q \xrightarrow[(0\ell_0, 1\ell_1)]{\tau} \nu \vec{y}, \vec{w}.(\text{id} | P' | Q')} \text{CLOSE-L} \\
\\
\frac{P \xrightarrow[\ell_0]{MN} \nu \vec{y}.(\text{id} | P') \quad Q \xrightarrow[\ell_1]{\overline{M}(\lambda)} \nu \vec{w}.(\{N/\lambda\} | Q') \quad \vec{w} \# P \quad \vec{y} \# Q, \vec{w}}{P | Q \xrightarrow[(0\ell_0, 1\ell_1)]{\tau} \nu \vec{y}, \vec{w}.(\text{id} | P' | Q')} \text{CLOSE-R}
\end{array}$$

■ **Figure 3** Rules for communication.

► **Definition 9** (locations). A location ℓ is of the form $s[t]$, where $s \in \{0, 1\}^*$ and $t \in \{0, 1\}^*$. If s or t is empty, we omit it (hence, we write $\epsilon[\epsilon]$ as $[\]$).

The colour coding above is to emphasise everywhere, what we call, the *location prefix* used to indicate the parallel component from which a transition originates, and the “ t part” of a location serves to identify which operand of the sum triggered the transition, as indicated in the SUM-L and SUM-R rules in Fig. 2.

To handle τ -transitions that originate from a synchronisation between an input and an output in two different locations, location labels, used to annotate transitions with events, may be pairs of locations, as employed in Fig. 3.

► **Definition 10** (location labels). A location label u is either a location ℓ or a pair of locations (ℓ_0, ℓ_1) , and we let $c(\ell_0, \ell_1) = (c\ell_0, c\ell_1)$ for $c \in \{0, 1\}$.

Events are pairs of action labels and location labels, as they appear above and below labelled transitions in Fig. 2 and 3. The role of the aliases and the usefulness of their prefix location is justified in Sec. 4.1 and 4.2. The rules SUM-L and SUM-R, and BANG rules have also been carefully designed, as explained in Sec. 4.3, 4.4 and 4.5. We will return, in those sections, to these rules to provide more detailed insight.

The only significant modification that we make to the standard syntax of applied π -calculus is that aliases have more structure than variables. Aliases, ranging over α, β, γ , are variables, extended with a, possibly empty, prefix of **0**s and **1**s representing the location of the process producing them. For convenience, we use λ, λ', \dots to range over variables used as aliases where the prefix is empty, and assume they are of a separate syntactic category from variables used for names and input binders, allowing us to drop some side conditions in rules.

We illustrate throughout the article how some transitions are derived. We sometimes apply the EXTRUDE or RES rules “in batch”, omit the id substitution, and list some hypothesis below the derivation. The derivation presented in Fig. 4 illustrates those conventions, but it also explicitly lists the occurrences of ϵ to help with readability. Indeed, this instance of ALIAS-OUT is trivial, since it turns the process on the left into an extended process with the identity substitution that records that nothing has yet been output.

Letting $P_{\text{ok}} = \nu m, n. (\bar{a}\langle m, n \rangle | m(x).[x = n]\overline{\text{ok}}\langle \text{ok} \rangle)$, we leave to the reader to convince themselves that the following transition is similarly derivable, using the PAR-L rule:

$$\text{id} | P_{\text{ok}} \xrightarrow[\mathbf{0}[\]]{\bar{a}(\mathbf{0}\lambda)} \nu m, n. (\{ \langle m, n \rangle / \mathbf{0}\lambda \} | \mathbf{0} | m(x).[x = n]\overline{\text{ok}}\langle \text{ok} \rangle).$$

$$\begin{array}{c}
\text{OUT} \\
\hline
\bar{c}\langle\langle m, n \rangle\rangle \xrightarrow[\epsilon[\epsilon]]{\bar{c}(\lambda)} \{\langle m, n \rangle / \lambda\} \mid 0 \quad n, m \# \text{fv}(\bar{c}(\lambda)) \\
\hline
\text{EXTRUDE } (\times 2) \\
\hline
\nu m. \nu n. \bar{c}\langle\langle m, n \rangle\rangle \xrightarrow[\epsilon[\epsilon]]{\bar{c}(\lambda)} \nu m. \nu n. (\{\langle m, n \rangle / \lambda\} \mid 0) \quad (\star) \\
\hline
\text{ALIAS-OUT} \\
\hline
\text{id} \mid (\nu m. \nu n. \bar{c}\langle\langle m, n \rangle\rangle) \xrightarrow[\epsilon[\epsilon]]{\bar{c}(\epsilon\lambda)} \nu m. \nu n. (\{\langle m, n \rangle / \lambda\} \mid 0) \\
\hline
m, n \# \text{ran}(\text{id}) = \emptyset \quad \text{fa}(c) = \emptyset \subseteq \text{dom}(\text{id}) = \emptyset \quad \epsilon\lambda \# \text{dom}(\text{id}) = \emptyset \quad (\star)
\end{array}$$

■ **Figure 4** First, simple example of derivation.

We discuss the next transition of P_{ok} in Sect. 4.1, and the dependence of the two events in Sect. 5. Another interesting example is given by the τ -transitions of the following process.

$$P_\tau = \nu z. ((\nu x. \bar{a}\langle\langle x, z \rangle\rangle \mid \nu y. \bar{b}\langle\langle y, z \rangle\rangle) \mid (a(x_1).P \mid b(x_2).Q))$$

This process can perform two different (and, as we will discuss in Sect. 5, independent) synchronizations whose location labels are $(00[], 10[])$ and $(01[], 11[])$. The following execution sequences illustrates how our semantics gracefully handles the two (parallel) sources of extrusions of the name z without any additional machinery.

$$\begin{array}{l}
\text{id} \mid P_\tau \xrightarrow[(00[], 10[])]{\tau} \nu z. \nu x. (\text{id} \mid (0 \mid \nu y. \bar{b}\langle\langle y, z \rangle\rangle) \mid (P\{\langle x, z \rangle / x_1\} \mid b(x_2).Q)) \\
\qquad \qquad \qquad \xrightarrow[(01[], 11[])]{\tau} \nu z. \nu x. \nu y. (\text{id} \mid (0 \mid 0) \mid (P\{\langle x, z \rangle / x_1\} \mid Q\{\langle y, z \rangle / x_2\}))
\end{array}$$

The full derivation trees for the above transitions are presented in Appendix A, Fig. 6. The derivation trees for the alternative sequence of transitions below are similar.

$$\begin{array}{l}
\text{id} \mid P_\tau \xrightarrow[(01[], 11[])]{\tau} \nu z. \nu y. (\text{id} \mid (\nu x. \bar{a}\langle\langle x, z \rangle\rangle \mid 0) \mid (a(x_1).P \mid Q\{\langle y, z \rangle / x_2\})) \\
\qquad \qquad \qquad \xrightarrow[(00[], 10[])]{\tau} \nu z. \nu y. \nu x. (\text{id} \mid (0 \mid 0) \mid (P\{\langle x, z \rangle / x_1\} \mid Q\{\langle y, z \rangle / x_2\}))
\end{array}$$

We reuse this process P_τ in Sect. 4.3 to illustrate the need for equivariance.

4.1 The Modern Applied π -Calculus Avoids Disjunctive Causality

The input and output prefixes $M(x).P$ and $\bar{M}\langle N \rangle.P$, respectively, indicate the channel as a message M , which is the modern approach to the applied π -calculus handling extruded messages [1]. Looking back at $\text{id} \mid P_{\text{ok}} \xrightarrow[\square]{\bar{a}(0\lambda)} \nu m, n. (\{\langle m, n \rangle / 0\lambda\} \mid 0 \mid m(x).[x = n]\text{ok}(\text{ok}))$, note that the active substitution $\{\langle m, n \rangle / 0\lambda\}$ can be used in subsequent events. For example, we may refer to the private name m by using the message $\text{fst}(0\lambda)$, which is then instantiated with the above active substitution, as illustrated by the following derivation of a transition (assuming a message theory featuring equation $\text{fst}\langle\langle m, n \rangle\rangle =_E m$):

Referring to channels that were extruded inside messages was not possible in early versions of the applied π -calculus [2]. This approach to extrusion is important to emphasise since adopting this modern approach significantly simplifies our labelled asynchronous transition system, as we explain next.

The key problem is to define a “stable” semantic in the presence of “link causality”. This means that, if multiple output events extrude the same name, we must record which output was used when that name appears in future events. An established approach to dealing with

$$\begin{array}{c}
 \frac{\text{fst}(\langle m, n \rangle) =_E m}{m(x).[x = n]\overline{\text{ok}}(\text{ok}) \xrightarrow[\square]{\text{fst}(\langle m, n \rangle) \text{snd}(\langle m, n \rangle)} \text{id} \mid [x = n]\overline{\text{ok}}(\text{ok}) \{ \text{snd}(\langle m, n \rangle) / x \}} \text{INP} \quad \epsilon \# 0 \\
 \frac{\quad}{0 \mid m(x).[x = n]\overline{\text{ok}}(\text{ok}) \xrightarrow[\square]{\text{fst}(\langle m, n \rangle) \text{snd}(\langle m, n \rangle)} \text{id} \mid 0 \mid [\text{snd}(\langle m, n \rangle) = n]\overline{\text{ok}}(\text{ok})} \text{PAR-R} \quad (*) \\
 \frac{\quad}{\sigma \mid 0 \mid m(x).[x = n]\overline{\text{ok}}(\text{ok}) \xrightarrow[\square]{\text{fst}(0\lambda) \text{snd}(0\lambda)} \sigma \mid 0 \mid [\text{snd}(\langle m, n \rangle) = n]\overline{\text{ok}}(\text{ok})} \text{ALIAS-FREE} \quad (**) \\
 \hline
 \nu m, n. (\sigma \mid 0 \mid m(x).[x = n]\overline{\text{ok}}(\text{ok})) \xrightarrow[\square]{\text{fst}(0\lambda) \text{snd}(0\lambda)} \nu m, n. (\sigma \mid 0 \mid [\text{snd}(\langle m, n \rangle) = n]\overline{\text{ok}}(\text{ok})) \text{EXT.}
 \end{array}$$

Letting $\sigma = \{ \langle m, n \rangle / 0\lambda \}$, since $\text{fst}(0\lambda) \text{snd}(0\lambda) \sigma = \text{fst}(\langle m, n \rangle) \text{snd}(\langle m, n \rangle)$,

$$\begin{array}{l}
 \epsilon \# \text{ran}(\sigma) = \{m, n\} \quad \text{fa}(\text{fst}(0\lambda) \text{snd}(0\lambda)) = \{0\lambda\} \subseteq \text{dom}(\sigma) = \{0\lambda\} \quad (*) \\
 n, m \# \text{fv}(\text{fst}(0\lambda) \text{snd}(0\lambda)) = \emptyset \quad (**)
 \end{array}$$

this “disjunctive dependency” is to extend the labels of transitions to record explicitly the set of output events each input depends on (by recording the source and target processes of the transition) [30, Def. 2.18]. Another established approach is to represent the disjunctive link causality in an “inclusive way” [20, p. 227], that “ensures that whenever an action with a bound subject is executed, at least one extrusion of that bound name must have been already executed”, but without recording which output was the *real* extruder that influenced another event. These additional mechanisms, used in related work, are used to acknowledge the difference between two extrusion events and recognise them as separate events.

The use of aliases avoids the disjunctive dependency problem entirely. Consider the process $\nu n. (\bar{a}\langle n \rangle \mid (\bar{a}\langle n \rangle \mid n(x).P))$ for example. This process can trigger both the output events $(\bar{a}(0\lambda), 0[\square])$ and $(\bar{a}(10\lambda), 10[\square])$:

$$\begin{aligned}
 \text{id} \mid \nu n. (\bar{a}\langle n \rangle \mid (\bar{a}\langle n \rangle \mid n(x).P)) &\xrightarrow[\square]{\bar{a}(0\lambda)} \nu n. (\{n/0\lambda\} \mid 0 \mid (\bar{a}\langle n \rangle \mid n(x).P)) \\
 &\xrightarrow[\square]{\bar{a}(10\lambda)} \nu n. (\{n/0\lambda\} \circ \{n/10\lambda\} \mid 0 \mid (0 \mid n(x).P))
 \end{aligned}$$

and, afterwards, only one of the input events $(0\lambda M, 11[\square])$ or $(10\lambda M, 11[\square])$ (letting $M' = M\{n/0\lambda\} \circ \{n/10\lambda\}$):

$$\nu n. (\{n/0\lambda\} \circ \{n/10\lambda\} \mid 0 \mid (0 \mid n(x).P)) \begin{cases} \xrightarrow[\square]{0\lambda M} \nu n. (\{n/0\lambda\} \circ \{n/10\lambda\} \mid 0 \mid (0 \mid P\{M'/x\})) \\ \xrightarrow[\square]{10\lambda M} \nu n. (\{n/0\lambda\} \circ \{n/10\lambda\} \mid 0 \mid (0 \mid P\{M'/x\})) \end{cases}$$

It is clear that input $0\lambda M$ is dependent on the output originating from the first thread in location 0 , while the input with alias $10\lambda M$ is dependent on the output in location 10 . Thus there is no need to perform event splitting, since there is no ambiguity about the source of the extrusion used to refer to channel n . The derivations of the transitions producing these events are presented in Appendix A.

4.2 Applied π -Calculus With Located Aliases

Recall that located aliases are variables prefixed with a string indicating the location in which the corresponding output occurred. The idea of prefixing aliases with a string representing a location is a novelty, which is necessary for the development of our LATS, as explained here. The strings themselves are, however, inherited from earlier work on LATS for CCS [39], where such strings are used to annotate labelled transitions and are used to determine whether actions are independent.

This modification to the syntax of the applied π -calculus serves two purposes: it enables us to define the independence relation only based on information from the events that label the transitions (Def. 12), and provides each location with a separate pool of aliases. Let us illustrate this latter purpose by picturing two execution sequences, with the locations omitted for now:

$$\text{id} \mid \nu m, n. (\bar{a}\langle m \rangle \mid \bar{a}\langle n \rangle) \begin{array}{l} \xrightarrow{\bar{a}\langle \alpha \rangle} \nu m, n. (\{m/\alpha\} \mid 0 \mid \bar{a}\langle n \rangle) \xrightarrow{\bar{a}\langle \beta \rangle} \nu m, n. (? \mid 0 \mid 0) \\ \xrightarrow{\bar{a}\langle \alpha \rangle} \nu m, n. (\{n/\alpha\} \mid \bar{a}\langle m \rangle \mid 0) \xrightarrow{\bar{a}\langle \beta \rangle} \nu m, n. (? \mid 0 \mid 0) \end{array} \quad (\text{rough diamond})$$

The challenge stems from needing to satisfy *diamond property 1*, as the two transitions are clearly independent: in the initial process $\nu m, n. (\bar{a}\langle m \rangle \mid \bar{a}\langle n \rangle)$, traditional semantics of the applied π -calculus allow us to use alias α for both initial transitions, whether the up or down transition is triggered. This freedom, unfortunately, violates *diamond property 1*, since taking the top or bottom transitions yields distinct substitutions, represented by ?. Of course, this difference in aliasing is irrelevant provided α and β are both fresh in every process involved, hence we “localise” the generation of fresh aliases.

To “localise” the generation of fresh aliases, we add a prefix to aliases, similarly to location labels, i.e., a (possibly empty) string of **1**s and **0**s, representing where an alias originates from within the binary tree of parallel locations. Those locations become genuine parts of the aliases, so that, e.g., **10** λ and **11** λ are distinct aliases (and of course **10** λ and **10** λ' are also distinct aliases, so there is an infinite supply of aliases for each location). Since our structural operational semantics does not assume that the parallel operator is commutative or associative, the bracketing of processes composed in parallel remains stable throughout an execution; and locations do not disappear as they would if we had included $P \mid 0 \equiv P$.

Going back to our example (rough diamond), this modification allows us to satisfy *diamond property 1* (ignoring the location label from under the arrows):

$$\text{id} \mid \nu m, n. (\bar{a}\langle m \rangle \mid \bar{a}\langle n \rangle) \begin{array}{l} \xrightarrow{\bar{a}\langle 0\lambda \rangle} \nu m, n. (\{m/0\lambda\} \mid 0 \mid \bar{a}\langle n \rangle) \xrightarrow{\bar{a}\langle 1\lambda \rangle} \nu m, n. (\{m/0\lambda\} \circ \{n/1\lambda\} \mid 0 \mid 0) \\ \xrightarrow{\bar{a}\langle 1\lambda \rangle} \nu m, n. (\{n/1\lambda\} \mid \bar{a}\langle m \rangle \mid 0) \xrightarrow{\bar{a}\langle 0\lambda \rangle} \nu m, n. (\{n/1\lambda\} \mid \bar{a}\langle m \rangle \mid 0) \end{array} \quad (\text{smooth diamond})$$

4.3 The Need for Equivariance

The order in which threads can be triggered can have an effect on the order of fresh name binders. Therefore, we consider states up to *equivariance*, that is, our structural congruence (Def. 8) extends α -equivalence such that $\nu x. \nu y. A \equiv \nu y. \nu x. A$. To see why we require equivariance, consider the following diamond of output transitions:

$$\begin{array}{c}
 \begin{array}{ccc}
 & \xrightarrow{\bar{a}(0\lambda)} & \nu m.\nu k.\langle\{m,k\}/_{0\lambda}\rangle \mid 0 \mid \nu n.\bar{a}\langle h(n)\rangle \\
 \uparrow & & \downarrow \\
 \nu m.\nu k.\bar{a}\langle\langle m,k\rangle\rangle \mid \nu n.\bar{a}\langle h(n)\rangle & & \nu m.\nu k.\nu n.\langle\{m,k\}/_{0\lambda}\rangle \circ \langle\{h(n)\}/_{1\lambda}\rangle \mid 0 \mid 0 \\
 & & \equiv \\
 & & \nu n.\nu m.\nu k.\langle\{h(n)\}/_{1\lambda}\rangle \circ \langle\{m,k\}/_{0\lambda}\rangle \mid 0 \mid 0 \\
 & & \uparrow \\
 & \xrightarrow{\bar{a}(1\lambda)} & \nu n.\langle\{h(n)\}/_{1\lambda}\rangle \mid \nu m.\nu k.\bar{a}\langle\langle m,k\rangle\rangle \mid 0
 \end{array} \\
 \bar{a}(0\lambda) & & \bar{a}(1\lambda) \\
 \bar{a}(1\lambda) & & \bar{a}(0\lambda)
 \end{array}$$

(Diamond upto equivariance)

Without taking the quotient, both transitions would reach different states, and any independence relation making the consecutive events $\bar{a}(0\lambda)$ and $\bar{a}(1\lambda)$ independent could not satisfy *diamond property 2*. This is also because the substitutions are the same function, hence the order in which composition is applied does not distinguish the extended processes.

Equivariance is also essential for identifying states that are the same after independent synchronisations. For example, the two execution sequences of $\text{id} \mid P_\tau$ discussed earlier would not reach equivalent states without equivariance. More explicitly, after two τ transitions $\text{id} \mid P_\tau$ can either reach the state $\nu z.\nu y.\nu x.(\text{id} \mid (0 \mid 0) \mid (P\langle\{x,z\}/_{x_1}\rangle \mid Q\langle\{y,z\}/_{x_2}\rangle))$ or the state $\nu z.\nu x.\nu y.(\text{id} \mid (0 \mid 0) \mid (P\langle\{x,z\}/_{x_1}\rangle \mid Q\langle\{y,z\}/_{x_2}\rangle))$ depending on which synchronisation is applied first. Observe how these extended processes only differ in that the binders for x and y are swapped, and hence are the same state up to equivariance.

4.4 Distinguishing Events in Conflict

The *event determinism* property of LATS requires more care in defining the semantics for the choice operator. Since event determinism is more fine-grained than the concept of “action determinism” in works on POR [5, Definition 4.1], our work is useful there too.

Our mechanism giving a located semantics to choice is similar to *proved transtions* [12] in the sense that our locations $s[t]$ contain information not only about the parallel structure (given by s), but also about the structure of choices (given by t). For instance, in transition

$$\text{id} \mid (((P_1 + a(x).P) + P_2) \mid (P_3 + \bar{a}\langle n\rangle)) \mid P_4 \xrightarrow[\mathbf{0}(0[01],1[1])]{\tau} \text{id} \mid (P\langle\{n\}/_x\rangle \mid 0) \mid P_4,$$

(derived in Fig. 5) the first *choice label* [01] indicates that $a(x).P$ was responsible for the input action in $(P_1 + a(x).P) + P_2$. Our choice labels diverge, however, from [39] where each location contains the source and target processes involved in that transition. We found that this established approach for CCS does not appear to work for applied π -calculus extended processes, such as $\nu m,n.(\{m\}/_{\lambda_0} \mid \bar{c}\langle\langle m,n\rangle\rangle + \bar{c}\langle\langle n,m\rangle\rangle)$, for which transitions $\xrightarrow[\overline{[\bar{c}\langle\langle m,n\rangle\rangle][0}]{\bar{c}(\lambda_1)}$ and $\xrightarrow[\overline{[\bar{c}\langle\langle n,m\rangle\rangle][0}]{\bar{c}(\lambda_1)}$ could be made by either branch of the choice upto equivariance, unless we add information that would break diamond properties.

Besides providing a more concise notation, our location labels unambiguously indicate which output in the non-deterministic choice was triggered in the following two co-initial transitions:

$$\nu m,n.(\{m\}/_{\lambda_0} \mid \bar{c}\langle\langle m,n\rangle\rangle + \bar{c}\langle\langle n,m\rangle\rangle) \left\{ \begin{array}{l} \xrightarrow[\mathbf{1}]{\bar{c}(\lambda_1)} \nu m,n.(\{m\}/_{\lambda_0} \circ \langle\{n,m\}/_{\lambda_1}\rangle \mid 0) \\ \xrightarrow[\mathbf{0}]{\bar{c}(\lambda_1)} \nu m,n.(\{m\}/_{\lambda_0} \circ \langle\{m,n\}/_{\lambda_1}\rangle \mid 0) \end{array} \right.$$

$$\begin{array}{c}
\frac{}{a(x).P \xrightarrow{a^n} \text{id} \mid P\{^n/x\}} \text{INP} \\
\frac{}{P_1 + a(x).P \xrightarrow{a^n} \text{id} \mid P\{^n/x\}} \text{SUM-R} \quad \frac{}{\bar{a}\langle n \rangle \xrightarrow{\bar{a}(\lambda)} \{^n/\lambda\} \mid 0} \text{OUT} \\
\frac{}{(P_1 + a(x).P) + P_2 \xrightarrow{a^n} \text{id} \mid P\{^n/x\}} \text{SUM-L} \quad \frac{}{P_3 + \bar{a}\langle n \rangle \xrightarrow{\bar{a}(\lambda)} \{^n/\lambda\} \mid 0} \text{SUM-R} \quad (*) \\
\frac{}{((P_1 + a(x).P) + P_2) \mid (P_3 + \bar{a}\langle n \rangle) \xrightarrow{\tau} \text{id} \mid P\{^n/x\} \mid 0} \text{CLOSE-R} \quad P_4 \# \emptyset \\
\frac{}{((P_1 + a(x).P) + P_2) \mid (P_3 + \bar{a}\langle n \rangle) \mid P_4 \xrightarrow{\tau} \text{id} \mid (P\{^n/x\} \mid 0) \mid P_4} \text{PAR-L} \quad (**) \\
\frac{}{\text{id} \mid ((P_1 + a(x).P) + P_2) \mid (P_3 + \bar{a}\langle n \rangle) \mid P_4 \xrightarrow{\tau} \text{id} \mid (P\{^n/x\} \mid 0) \mid P_4} \text{ALIAS-FREE} \\
\epsilon \# (P_1 + a(x).P) + P_2 \quad \epsilon \# P_3 + \bar{a}\langle n \rangle, \emptyset \quad (*) \\
\epsilon \# \text{ran}(\text{id}) = \emptyset \quad \text{fa}(\tau) = \emptyset \subseteq \text{dom}(\text{id}) = \emptyset \quad (**)
\end{array}$$

■ **Figure 5** Derivation example involving sum and synchronisation.

Moreover, identifying the events would violate *event determinism*, as the resulting extended processes are different. Notice that we do not record name binders in the events, since, unlike CSS, names move around in a way that would violate diamonds. Instead, name binders are handled by mechanisms used inside the proofs of diamond properties.

4.5 Addressing the Location of Replicated Processes

As usual, the replication operator $!$ is used to represent an unbounded number of sessions. Recording more structure than the labels when defining events prevents the rule BANG from creating image-finiteness problems that the same rule creates for an action-based LTS. Indeed, since we have *event determinism*, the image of any extended process and event is a singleton upto structural congruence. This design decision gives to every replicated process an infinite pool of explicit location names, and allows each of these locations to be triggered in any order. This is important to satisfy *diamond property 2*.

To explain this, consider the following example that use the process $\bar{a}\langle y \rangle . \bar{b}\langle z \rangle . Q = P_b$, which can perform the following transitions:

$$\text{id} \mid !P_b \xrightarrow{\bar{a}(0\lambda)} \{y/0\lambda\} \mid \bar{b}\langle z \rangle . Q \mid !P_b \xrightarrow{\bar{b}(0\lambda')} \{y/0\lambda\} \circ \{z/0\lambda'\} \mid Q \mid !P_b$$

Our definition of independence relation (Def. 11) – and, we believe, *any* reasonable location-based definition of independence – would ensure that the two events $(\bar{a}(0\lambda), 0\llbracket \rrbracket)$ and $(\bar{b}(0\lambda'), 0\llbracket \rrbracket)$ are not independent – they are treated as coming from *the same* location, even if that location “did not exist” when P_b started its execution. Hence *diamond property 2* cannot apply and these events cannot permute, as expected. This mechanism echoes e.g., the dependency relation that can be developed to accommodate replication for CCS [23].

In contrast, consider the following transitions, also originating from the same process $!P_b$:

$$\begin{array}{c}
\bar{a}(0\lambda) \xrightarrow{0\llbracket \rrbracket} \{y/0\lambda\} \mid \bar{b}\langle z \rangle . Q \mid !P_b \xrightarrow{\bar{a}(10\lambda)} \{y/10\lambda\} \mid \bar{b}\langle z \rangle . Q \mid !P_b \\
\bar{a}(10\lambda) \xrightarrow{10\llbracket \rrbracket} \{y/10\lambda\} \mid P_b \mid (\bar{b}\langle z \rangle . Q \mid !P_b) \xrightarrow{\bar{a}(0\lambda)} \{y/0\lambda\} \mid \bar{b}\langle z \rangle . Q \mid !P_b
\end{array}$$

(Diamond with a bang)

The events on the left-hand side of the diamond above are expected to be independent. Fortunately, by triggering BANG twice, we can permute these transitions as required by *diamond property 2*. That is, the right side of the above diamond exists.

An alternative solution could be to use explicit names to label locations, and partially ordering those labels to reflect the hierarchy of locations, and then minting fresh names for each transition of a replication [13, 44]. Using opaque names as locations, however, would have forced us to record them in the syntax of processes, that would have become e.g., of the form $\ell_1 :: \bar{b}\langle z \rangle.Q \mid (\ell_2 :: \bar{b}\langle z \rangle.Q \mid !P)$ [12]. The intent is however the same.

5 The Independence Relation and the Main Result

In this section, we define what it means for two events to be independent. To do this, firstly, we define structural independence, which ensures that two events occur in different locations by checking that it is not the case that one location prefix is a prefix (as a string) of the other event's location prefix.

► **Definition 11** (structural independence). *Define \mathcal{Loc} on location labels such that $\mathcal{Loc}(\ell) = \{\ell\}$ and $\mathcal{Loc}(\ell_0, \ell_1) = \{\ell_0, \ell_1\}$. The structural independence relation I_ℓ on location labels is the least relation defined by $u_0 I_\ell u_1$ whenever for all locations $\ell_0 \in \mathcal{Loc}(u_0)$ and $\ell_1 \in \mathcal{Loc}(u_1)$, there exist a string $s \in \{0, 1\}^*$ and locations ℓ'_0, ℓ'_1 , such that either: $\ell_0 = s0\ell'_0$ and $\ell_1 = s1\ell'_1$; or $\ell_0 = s1\ell'_0$ and $\ell_1 = s0\ell'_1$.*

For example, consider the locations of the four output events in $\bar{a}\langle a \rangle \mid \bar{b}\langle b \rangle.(\bar{c}\langle c \rangle \mid \bar{d}\langle d \rangle)$. The output on channel a (location prefix 0) is structurally independent from all other outputs. The output on channel b (location prefix 1) is not structurally independent with respect to the outputs on channels c or d ; which will have location prefixes 10 and 11 respectively, both with 1 as a common prefix. However, the outputs on channel c and d are independent, since neither 10 nor 11 are prefixes of each other.

Independence on events in addition detects whether an output influences another action. That is, in addition to structural independence, we have link independence.

► **Definition 12** (independence of events). *Events $e = (\pi, u)$ are pairs of labels π and location labels u . The independence relation I on events is the least symmetric relation such that $(\pi_0, u_0) I (\pi_1, u_1)$ whenever $u_0 I_\ell u_1$ and if $\pi_0 = \bar{M}(\alpha)$, then $\alpha \# \pi_1$.*

Remember that P_{ok} from Sect. 4 and 4.1 can trigger the event $(\bar{a}(0\lambda), 0)$ followed by $(fst(0\lambda) snd(0\lambda), 1)$. Even if the locations are independent (as $0 I_\ell 1$), the two events are not independent, since 0λ is not fresh in $(fst(0\lambda) snd(0\lambda))$.

► **Theorem 13.** *The structural operational semantics in Fig. 2 and 3 generates a labelled asynchronous transition system with respect to the independence relation I from Def. 12, i.e., it respects Def. 1 – 3, where events are the pairs of action and location labels (π, u) , as they appear on the labels of transitions, and states are extended processes modulo the structural congruence from Def. 8. [see proof in Appendix B]*

Link independence is only required to permute an output event followed by an independent event, when establishing *diamond property 2*. The main intricacy compared to CCS is to ensure that the name restrictions occurring along any common prefix of two independent transitions are handled correctly – this is how the parallel extrusion problem in related theories manifests itself in this theory. We handle this problem entirely within the proof, via variables accumulated in a function that picks out the component of a process corresponding to a location prefix, rather than within the semantics as in related work [30, 50].

6 Related Work

The earliest papers on partial-order reduction for security are not working with the applied π -calculus, but rely on constructing execution DAGs (or Mazurkiewicz traces) recording all input-output dependencies; as a result, the protocols considered in [18, 21, 27, 38] are essentially threads of inputs and outputs, disregarding channels.

Our paper is closer to more recent approaches to POR for the applied π -calculus [6, 7, 8]. However, one limitation of these works is that they require processes to be of a particular form. In that related line of work, structural independence is based on the channel of an input and output action, thus considering events structurally independent only when they employ distinct channels. Thus, their scope is restricted to processes in which every location is a single thread of sequential actions (e.g., cannot spawn parallel threads in a location) and **if-then-else** branches that employ a unique channel. While many finite protocol problems can be formulated with a fixed number of threads, each employing separate channels, this is still a significant restriction. In contrast, we base the structural independence on the components' addresses, which allows us to consider the “full” applied π -calculus.

For modelling infinite protocols (or protocols where the same, or multiple, actors can engage in multiple parallel sessions) one normally uses replication, and thus parallel extrusion appears naturally. Some of the above related works [7] approximate replication by creating fresh channels manufactured every time a parallel component is created. However, since these works do not have mechanisms for dealing with disjunctive causality (which in these settings is required because of the use of a different mechanism for extrusion, similar to the standard π -calculus), channel extrusion is only supported for processes where parallel extrusion of channels never occurs. In contrast, our approach, where channels are extruded like any other message and aliases resolve the aforementioned disjunctive causality problem, supports all forms of processes including replication and parallel extrusion.

A further difference compared to the above work concerns τ transitions. In the related work we are discussing, there is not enough information to determine whether two τ transitions are independent, and hence such parts of a protocol's behaviour cannot be considered for POR optimisation. In our semantics, we resolve this problem using our structural independence relation based on the recorded locations responsible for the input and output actions involved in a τ transition. Our solution simply lifts classic work on CCS [39] to this security setting, while taking care to handle parallel extrusion correctly (recall the P_τ example from Sect. 4).

All the above work does not follow the non-interleaving tradition in the sense that they do not feature *diamond property 1*. It is possible in the semantics of [6, 8, 17] that two transitions that we deem independent and are enabled from some process will disable each other, since their executions change the substitution on which the other depends. This is the key problem our located aliases address. The only diamond property that related approaches obtain (and for a limited subset of processes, as explained above) is the “reordering of sequential independent transitions”, which is our *diamond property 2*. However, this property alone can be achieved without located aliases since, anyway, two sequential outputs would be named differently by the regular constraints of the applied π -calculus, which simply ensure that an extruded alias is globally fresh.

Event determinism can be achieved in works such as [7, 8] only for choice of the type **if-then-else**. By borrowing from proved transitions [12], our events record the precise branch of a general non-deterministic choice from which they originate, thereby achieving event-determinism for all types of processes. This problem has been acknowledged [6], and an

alternative approach to POR using *sleep sets* has been proposed. This approach essentially defines independence in terms of events satisfying our *diamond property 1*, but since the semantics they employ allows two concurrently enabled outputs on the same channel to use the same label, such events would incorrectly be considered not independent.

We see a good opportunity in adopting (concepts from) our semantics in the settings and tools of the above papers. This possibility has been one of our goals all along, and guided our decisions to consider a syntax very close to the standard applied π -calculus and to define a structural operational semantics that just extends previous semantics for applied π [32]. Even the choice of LATS was guided by our wish to stay within the realm of transition systems, yet to go over from interleaving to non-interleaving semantics. Hence, we expect it to be possible to upgrade tools from the above mentioned papers, so that they may fully support POR for all processes.

7 Conclusion

The work we build on incorporates elements of the modern applied π -calculus [1] – aliases for extrusion of channels as messages – into a structural operational semantics [31, 32]. Our semantics in Fig. 2 and 3 transforms this established semantics into a non-interleaving structural operational semantics by recording on transitions also the location from which an event originates as well as the location from where an alias representing an output originates. The former is a standard device, coming from non-interleaving semantics for CCS [12, 39] and π -calculus [30], while the latter “located alias” is the key technical innovation required to ensure that our structural operational semantics defines a LATS (Theorem 13). In this way we obtain a genuine operational semantics, with a remarkably simple independence relation (Def. 12) for such a powerful calculus. Moreover, this paper can also be seen as proposing LATS as the semantic objects for applied π -calculi, instead of transition systems, so as to bridge other non-interleaving models to which LATS have been related in the literature.

Because LATS have been shown [28] to be exactly the higher-dimensional automata of dimension 2, we can reuse the definitions for higher-dimensional automata of the classical concurrency bisimulations (i.e., of ST-, history preserving-, and hereditary history preserving-bisimulations) for LATS [47]. Moreover, through relations of LATS with configuration structures [49] and event structures [51] we can reuse also other concurrency bisimulations [48]. In related work, partial-order semantics have been employed in tools for optimising verification of equivalence properties [17]. Making precise the connection between the equivalences employed in such tools and the above non-interleaving equivalences for LATS is future work.

References

- 1 Martín Abadi, Bruno Blanchet, and Cédric Fournet. The applied pi calculus: Mobile values, new names, and secure communication. *J. ACM*, 65(1):1:1–1:41, 2018. doi:10.1145/3127586.
- 2 Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL’01)*, pages 104–115, 2001. doi:10.1145/360204.360213.
- 3 Reynald Affeldt and Naoki Kobayashi. Partial order reduction for verification of spatial properties of pi-calculus processes. *Electron. Notes Theor. Comput. Sci.*, 128(2):151–168, 2004. Proceedings of the 11th International Workshop on Expressiveness in Concurrency (EXPRESS 2004). doi:10.1016/j.entcs.2004.11.034.
- 4 Myrto Arapinis, Tom Chothia, Eike Ritter, and Mark Ryan. Analysing unlinkability and anonymity using the applied pi calculus. In *23rd IEEE Computer Security Foundations Symposium (CSF’10)*, pages 107–121, 2010. doi:10.1109/CSF.2010.15.

- 5 David Baelde. *Contributions à la Vérification des Protocoles Cryptographiques*. Habilitation à diriger des recherches, Université Paris-Saclay, February 2021. URL: http://www.lsv.fr/~baelde/hdr/habilitation_baelde.pdf.
- 6 David Baelde, Stéphanie Delaune, and Lucca Hirschi. POR for security protocol equivalences – beyond action-determinism. In Javier López, Jianying Zhou, and Miguel Soriano, editors, *Computer Security – 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, September 3-7, 2018, Proceedings, Part I*, volume 11098 of *LNCS*, pages 385–405. Springer, 2018. doi:10.1007/978-3-319-99073-6_19.
- 7 David Baelde, Stéphanie Delaune, and Lucca Hirschi. Partial order reduction for security protocols. In Luca Aceto and David de Frutos-Escrig, editors, *26th International Conference on Concurrency Theory, CONCUR 2015, Madrid, Spain, September 1-4, 2015*, volume 42 of *LIPICs*, pages 497–510. Schloss Dagstuhl, 2015. doi:10.4230/LIPICs.CONCUR.2015.497.
- 8 David Baelde, Stéphanie Delaune, and Lucca Hirschi. A reduced semantics for deciding trace equivalence. *Log. Meth. Comput. Sci.*, 13(2), 2017. doi:10.23638/LMCS-13(2:8)2017.
- 9 Marek A. Bednarczyk. *Categories of asynchronous systems*. PhD thesis, University of Sussex, Brighton, UK, 1988.
- 10 Bruno Blanchet. Modeling and verifying security protocols with the applied pi calculus and proverif. *Foundations and Trends in Privacy and Security*, 1(1-2):1–135, 2016. doi:10.1561/33000000004.
- 11 Michele Boreale and Davide Sangiorgi. A fully abstract semantics for causality in the π -calculus. *Acta Inform.*, 35(5):353–400, 1998.
- 12 Gérard Boudol and Ilaria Castellani. A non-interleaving semantics for CCS based on proved transitions. *Fund. Inform.*, 11:433–452, 1988.
- 13 Gérard Boudol, Ilaria Castellani, Matthew Hennessy, and Astrid Kiehn. Observing localities. *Theor. Comput. Sci.*, 114(1):31–61, 1993. doi:10.1016/0304-3975(93)90152-J.
- 14 Sergiu Bursuc, Christian Johansen, and Shiwei Xu. Automated verification of dynamic root of trust protocols. In Matteo Maffei and Mark Ryan, editors, *International Conference on Principles of Security and Trust*, volume 10204 of *LNCS*, pages 95–116. Springer, 2017. doi:10.1007/978-3-662-54455-6_5.
- 15 Georgia Carabetta, Pierpaolo Degano, and Fabio Gadducci. CCS semantics via proved transition systems and rewriting logic. In Claude Kirchner and Hélène Kirchner, editors, *1998 International Workshop on Rewriting Logic and its Applications, WRLA 1998, Abbaye des Prémontrés at Pont-à-Mousson, France, September 1998*, volume 15 of *Electron. Notes Theor. Comput. Sci.*, pages 369–387. Elsevier, 1998. doi:10.1016/S1571-0661(05)80023-4.
- 16 Rohit Chadha, Vincent Cheval, Ștefan Ciobăcă, and Steve Kremer. Automated verification of equivalence properties of cryptographic protocols. *ACM Trans. Comput. Log.*, 17(4):23:1–23:32, September 2016. doi:10.1145/2926715.
- 17 Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina. Exploiting symmetries when proving equivalence properties for security protocols. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 905–922. ACM, 2019. doi:10.1145/3319535.3354260.
- 18 Edmund M. Clarke, Somesh Jha, and Wilfredo R. Marrero. Efficient verification of security protocols using partial-order reductions. *Int. J. Softw. Tools Technol. Transf.*, 4(2):173–188, 2003. doi:10.1007/s10009-002-0103-4.
- 19 Véronique Cortier, Antoine Dallon, and Stéphanie Delaune. SAT-Equiv: An efficient tool for equivalence properties. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, pages 481–494, August 2017. doi:10.1109/CSF.2017.15.
- 20 Silvia Crafa, Daniele Varacca, and Nobuko Yoshida. Event structure semantics of parallel extrusion in the pi-calculus. In Lars Birkedal, editor, *Foundations of Software Science and Computational Structures – 15th International Conference, FOSSACS 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 – April 1, 2012. Proceedings*, volume 7213 of *LNCS*, pages 225–239. Springer, 2012. doi:10.1007/978-3-642-28729-9_15.

- 21 Cas J. F. Cremers and Sjouke Mauw. Checking secrecy by means of partial order reduction. In Daniel Amyot and Alan W. Williams, editors, *System Analysis and Modeling, 4th International SDL and MSC Workshop, SAM 2004, Ottawa, Canada, June 1-4, 2004, Revised Selected Papers*, volume 3319 of *LNCS*, pages 171–188. Springer, 2004. doi:10.1007/978-3-540-31810-1_12.
- 22 Ioana Cristescu, Jean Krivine, and Daniele Varacca. Rigid families for CCS and the π -calculus. In Martin Leucker, Camilo Rueda, and Frank D. Valencia, editors, *Theoretical Aspects of Computing – ICTAC 2015 – 12th International Colloquium Cali, Colombia, October 29-31, 2015, Proceedings*, volume 9399 of *LNCS*, pages 223–240. Springer, 2015. doi:10.1007/978-3-319-25150-9_14.
- 23 Pierpaolo Degano, Fabio Gadducci, and Corrado Priami. Causality and replication in concurrent processes. In Manfred Broy and Alexandre V. Zamulin, editors, *Perspectives of Systems Informatics, 5th International Andrei Ershov Memorial Conference, PSI 2003, Akademgorodok, Novosibirsk, Russia, July 9-12, 2003, Revised Papers*, volume 2890 of *LNCS*, pages 307–318. Springer, 2003. doi:10.1007/978-3-540-39866-0_30.
- 24 Pierpaolo Degano and Corrado Priami. Proved trees. In Werner Kuich, editor, *Automata, Languages and Programming, 19th International Colloquium, ICALP92, Vienna, Austria, July 13-17, 1992, Proceedings*, volume 623 of *LNCS*, pages 629–640. Springer, 1992. doi:10.1007/3-540-55719-9_110.
- 25 Pierpaolo Degano and Corrado Priami. Non-interleaving semantics for mobile processes. *Theor. Comput. Sci.*, 216(1-2):237–270, 1999. doi:10.1016/S0304-3975(99)80003-6.
- 26 Pierpaolo Degano and Corrado Priami. Enhanced operational semantics. *ACM Comput. Surv.*, 33(2):135–176, 2001. doi:10.1145/384192.384194.
- 27 Wan Fokkink, Mohammad Torabi Dashti, and Anton Wijs. Partial order reduction for branching security protocols. In *2010 10th International Conference on Application of Concurrency to System Design*, pages 191–200. IEEE, 2010. doi:10.1109/ACSD.2010.19.
- 28 Éric Goubault. Labelled cubical sets and asynchronous transition systems: an adjunction. In *Preliminary Proceedings CMCIM'02*, 2002. URL: <http://www.lix.polytechnique.fr/~goubault/papers/cmcm02.ps.gz>.
- 29 Thomas Troels Hildebrandt. *Categorical Models for Concurrency: Independence, Fairness and Dataflow*. PhD thesis, BRICS, University of Aarhus, February 2000. URL: <http://www.brics.dk/DS/00/1/>.
- 30 Thomas Troels Hildebrandt, Christian Johansen, and Håkon Normann. A stable non-interleaving early operational semantics for the pi-calculus. *J. Log. Algebr. Methods Program.*, 104:227–253, 2019. doi:10.1016/j.jlamp.2019.02.006.
- 31 Ross Horne and Sjouke Mauw. Discovering epassport vulnerabilities using bisimilarity. *Log. Meth. Comput. Sci.*, 17(2):24, 2021. doi:10.23638/LMCS-17(2:24)2021.
- 32 Ross Horne, Sjouke Mauw, and Semen Yurkov. Compositional analysis of protocol equivalence in the applied π -calculus using quasi-open bisimilarity. In Antonio Cerone and Peter Csaba Ölveczky, editors, *Theoretical Aspects of Computing – ICTAC 2021 – 18th International Colloquium, Virtual Event, Nur-Sultan, Kazakhstan, September 8-10, 2021, Proceedings*, volume 12819 of *LNCS*, pages 235–255. Springer, 2021. doi:10.1007/978-3-030-85315-0_14.
- 33 Ross Horne, Sjouke Mauw, and Semen Yurkov. Unlinkability of an improved key agreement protocol for emv 2nd gen payments. In Stefano Calzavara, editor, *IEEE Computer Security Foundations Symposium 2022 (CSF2022), August, 2022, Haifa, Israel, 2022*. to appear. URL: <https://satoss.uni.lu/members/ross/pdf/emv.pdf>.
- 34 Lalita Jategaonkar Jagadeesan and Radha Jagadeesan. Causality and true concurrency: A data-flow analysis of the pi-calculus. In *International Conference on Algebraic Methodology and Software Technology*, pages 277–291. Springer, 1995.
- 35 Steve Kremer and Robert Künnemann. Automated analysis of security protocols with global state. *JCS*, 24(5):583–616, 2016. doi:10.3233/JCS-160556.
- 36 Steve Kremer and Mark Ryan. Analysis of an electronic voting protocol in the applied pi calculus. In Mooly Sagiv, editor, *Programming Languages and Systems: 14th European Symposium on Programming (ESOP'05 at ETAPS'05)*, volume 3444 of *LNCS*, pages 186–200. Springer-Verlag, 2005. doi:10.1007/978-3-540-31987-0_14.

- 37 Ivan Lanese, Iain C. C. Phillips, and Irek Ulidowski. An axiomatic approach to reversible computation. In Jean Goubault-Larrecq and Barbara König, editors, *Foundations of Software Science and Computation Structures – 23rd International Conference, FOSSACS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings*, volume 12077 of *LNCS*, pages 442–461. Springer, 2020. doi:10.1007/978-3-030-45231-5_23.
- 38 Sebastian Mödersheim, Luca Viganò, and David Basin. Constraint differentiation: Search-space reduction for the constraint-based analysis of security protocols. *J. Comput. Secur.*, 18(4):575–618, 2010. doi:10.3233/JCS-2009-0351.
- 39 Madhavan Mukund and Mogens Nielsen. CCS, Location and Asynchronous Transition Systems. In R. K. Shyamasundar, editor, *Foundations of Software Technology and Theoretical Computer Science, 12th Conference, New Delhi, India, December 18-20, 1992, Proceedings*, volume 652 of *LNCS*, pages 328–341. Springer, 1992. doi:10.1007/3-540-56287-7_116.
- 40 Kirstin Peters, Jens-Wolfhard Schicke-Uffmann, Ursula Goltz, and Uwe Nestmann. Synchrony versus causality in distributed systems. *Math. Struct. Comput. Sci.*, 26(8):1459–1498, 2016. doi:10.1017/S0960129514000644.
- 41 Carl Adam Petri. Fundamentals of a theory of asynchronous information flow. In *Information Processing, Proceedings of the 2nd IFIP Congress 1962, Munich, Germany, August 27 – September 1, 1962*, pages 386–390. North-Holland, 1962.
- 42 Iain Phillips and Irek Ulidowski. Reversibility and models for concurrency. *Electron. Notes Theor. Comput. Sci.*, 192(1):93–108, 2007. doi:10.1016/j.entcs.2007.08.018.
- 43 Mark Dermot Ryan and Ben Smyth. Applied pi calculus. In Véronique Cortier and Steve Kremer, editors, *Formal Models and Techniques for Analyzing Security Protocols*, volume 5 of *Cryptology and Information Security Series*, pages 112–142. IOS Press, 2011. doi:10.3233/978-1-60750-714-7-112.
- 44 Davide Sangiorgi. A theory of bisimulation for the pi-calculus. *Acta Inform.*, 33(1):69–97, 1996. doi:10.1007/s002360050036.
- 45 Michael W. Shields. Concurrent machines. *Comput. J.*, 28(5):449–465, 1985. doi:10.1093/comjnl/28.5.449.
- 46 Alwen Tiu, Nam Nguyen, and Ross Horne. Spec: An equivalence checker for security protocols. In Atsushi Igarashi, editor, *14th Asian Symposium on Programming Languages and Systems (APLAS’16)*, volume 10017 of *LNCS*, pages 87–95. Springer, 2016. doi:10.1007/978-3-319-47958-3_5.
- 47 Rob van Glabbeek. On the Expressiveness of Higher Dimensional Automata. *Theor. Comput. Sci.*, 356(3):265–290, 2006.
- 48 Rob van Glabbeek and Ursula Goltz. Refinement of actions and equivalence notions for concurrent systems. *Acta Inform.*, 37(4/5):229–327, 2001. doi:10.1007/s002360000041.
- 49 Rob van Glabbeek and Gordon D. Plotkin. Configuration structures, event structures and petri nets. *Theor. Comput. Sci.*, 410(41):4111–4159, 2009. doi:10.1016/j.tcs.2009.06.014.
- 50 Daniele Varacca and Nobuko Yoshida. Typed event structures and the linear pi-calculus. *Theor. Comput. Sci.*, 411(19):1949–1973, 2010. doi:10.1016/j.tcs.2010.01.024.
- 51 Glynn Winskel and Mogens Nielsen. Models for concurrency. In Samson Abramsky, Dov M. Gabbay, and Thomas Stephen Edward Maibaum, editors, *Semantic Modelling*, volume 4 of *Handbook of Logic in Computer Science*, pages 1–148. Oxford University Press, 1995.

A Derivations for selected example transitions

We present the derivations of the transitions used to illustrate stability in Sect. 4.1. We are explicit, in the input transitions, about the source of the extruded name.

$$\begin{array}{c}
 \frac{}{\bar{a}(n) \xrightarrow[\perp]{\bar{a}(\lambda)} \{n/\lambda\} \mid 0} \text{OUT} \quad \epsilon \# \bar{a}(n) \mid n(x).P \\
 \frac{}{\bar{a}(n) \mid (\bar{a}(n) \mid n(x).P) \xrightarrow[\perp]{\bar{a}(\lambda)} \{n/\lambda\} \mid 0 \mid (\bar{a}(n) \mid n(x).P)} \text{PAR-L} \quad n \# \text{fv}(\bar{a}(\lambda)) \\
 \frac{}{\nu n.(\bar{a}(n) \mid (\bar{a}(n) \mid n(x).P)) \xrightarrow[\perp]{\bar{a}(\lambda)} \nu n.(\{n/\lambda\} \mid 0 \mid (\bar{a}(n) \mid n(x).P))} \text{EXTRUDE} \quad (\star) \\
 \frac{}{\text{id} \mid \nu n.(\bar{a}(n) \mid (\bar{a}(n) \mid n(x).P)) \xrightarrow[\perp]{\bar{a}(0\lambda)} \nu n.(\{n/0\lambda\} \mid 0 \mid (\bar{a}(n) \mid n(x).P))} \text{ALIAS-OUT} \\
 n \# \text{ran}(\text{id}) = \emptyset \quad \text{fa}(a) = \emptyset \subseteq \text{dom}(\text{id}) = \emptyset \quad 0\lambda \# \text{dom}(\text{id}) = \emptyset \quad (\star)
 \end{array}$$

$$\begin{array}{c}
 \frac{}{\bar{a}(n) \xrightarrow[\perp]{\bar{a}(\lambda)} \{n/\lambda\} \mid 0} \text{OUT} \quad \epsilon \# n(x).P \\
 \frac{}{\bar{a}(n) \mid n(x).P \xrightarrow[\perp]{\bar{a}(\lambda)} \{n/\lambda\} \mid 0 \mid n(x).P} \text{PAR-L} \quad \epsilon \# 0 \\
 \frac{}{0 \mid (\bar{a}(n) \mid n(x).P) \xrightarrow[\perp]{\bar{a}(\lambda)} \{n/\lambda\} \mid 0 \mid (0 \mid n(x).P)} \text{PAR-R} \quad (\star) \\
 \frac{}{\{n/0\lambda\} \mid 0 \mid (\bar{a}(n) \mid n(x).P) \xrightarrow[\perp]{\bar{a}(10\lambda)} \{n/0\lambda\} \circ \{n/10\lambda\} \mid 0 \mid (\bar{a}(n) \mid n(x).P)} \text{ALIAS-OUT} \quad n \# \text{fv}(\bar{a}(10\lambda)) \\
 \frac{}{\nu n.(\{n/0\lambda\} \mid 0 \mid (\bar{a}(n) \mid n(x).P)) \xrightarrow[\perp]{\bar{a}(10\lambda)} \nu n.(\{n/0\lambda\} \circ \{n/10\lambda\} \mid 0 \mid (0 \mid n(x).P))} \text{EXTRUDE} \\
 \epsilon \# \text{ran}(\{n/0\lambda\}) = n \quad \text{fa}(a) = \emptyset \subseteq \text{dom}(\{n/0\lambda\}) = 0\lambda \quad 10\lambda \# \text{dom}(\{n/0\lambda\}) = 0\lambda \quad (\star)
 \end{array}$$

$$\begin{array}{c}
 \frac{}{n(x).P \xrightarrow[\perp]{nM'} \text{id} \mid P\{M'/x\}} \text{INP} \quad \epsilon \# 0 \\
 \frac{}{0 \mid n(x).P \xrightarrow[\perp]{nM'} \text{id} \mid 0 \mid P\{M'/x\}} \text{PAR-R} \quad \epsilon \# 0 \\
 \frac{}{0 \mid (0 \mid n(x).P) \xrightarrow[\perp]{nM'} \text{id} \mid 0 \mid (0 \mid P\{M'/x\})} \text{PAR-R} \quad (\star) \\
 \frac{}{\{n/0\lambda\} \circ \{n/10\lambda\} \mid 0 \mid (0 \mid n(x).P) \xrightarrow[\perp]{0\lambda M} \{n/0\lambda\} \circ \{n/10\lambda\} \mid 0 \mid (0 \mid P\{M'/x\})} \text{ALIAS-FREE} \quad n \# \text{fv}(0\lambda M) \\
 \frac{}{\nu n.(\{n/0\lambda\} \circ \{n/10\lambda\} \mid 0 \mid (0 \mid n(x).P)) \xrightarrow[\perp]{0\lambda M} \nu n.(\{n/0\lambda\} \circ \{n/10\lambda\} \mid 0 \mid (0 \mid P\{M'/x\}))} \text{RES}
 \end{array}$$

$$\text{Letting } \sigma = \{n/0\lambda\} \circ \{n/10\lambda\} \text{ and } (0\lambda M)\sigma = nM', \\
 \epsilon \# \text{ran}(\sigma) = \{n\} \quad \text{fa}(0\lambda M) = \{0\lambda\} \subseteq \text{dom}(\sigma) = \{0\lambda, 10\lambda\} \quad (\star)$$

$$\begin{array}{c}
 \frac{}{n(x).P \xrightarrow[\perp]{nM'} \text{id} \mid P\{M'/x\}} \text{INP} \quad \epsilon \# 0 \\
 \frac{}{0 \mid n(x).P \xrightarrow[\perp]{nM'} \text{id} \mid 0 \mid P\{M'/x\}} \text{PAR-R} \quad \epsilon \# 0 \\
 \frac{}{0 \mid (0 \mid n(x).P) \xrightarrow[\perp]{nM'} \text{id} \mid 0 \mid (0 \mid P\{M'/x\})} \text{PAR-R} \quad (\star) \\
 \frac{}{\{n/0\lambda\} \circ \{n/10\lambda\} \mid 0 \mid (0 \mid n(x).P) \xrightarrow[\perp]{10\lambda M} \{n/0\lambda\} \circ \{n/10\lambda\} \mid 0 \mid (0 \mid P\{M'/x\})} \text{ALIAS-FREE} \quad n \# \text{fv}(10\lambda M) \\
 \frac{}{\nu n.(\{n/0\lambda\} \circ \{n/10\lambda\} \mid 0 \mid (0 \mid n(x).P)) \xrightarrow[\perp]{10\lambda M} \nu n.(\{n/0\lambda\} \circ \{n/10\lambda\} \mid 0 \mid (0 \mid P\{M'/x\}))} \text{RES}
 \end{array}$$

$$\text{Letting } \sigma = \{n/0\lambda\} \circ \{n/10\lambda\} \text{ and } (10\lambda M)\sigma = nM', \\
 \epsilon \# \text{ran}(\sigma) = \{n\} \quad \text{fa}(10\lambda M) = \{10\lambda\} \subseteq \text{dom}(\sigma) = \{0\lambda, 10\lambda\} \quad (\star)$$

$$\begin{array}{c}
\frac{\text{OUT}}{\bar{a}\langle(x,z)\rangle \xrightarrow{\bar{a}(\lambda)} \{\langle(x,z)/\lambda\rangle \mid 0\}} \quad x \# \text{fv}(\bar{a}(\lambda)) \\
\frac{\text{EXTRUDE}}{\nu x.\bar{a}\langle(x,z)\rangle \xrightarrow{\bar{a}(\lambda)} \nu x.\{\langle(x,z)/\lambda\rangle \mid 0\}} \quad x \# \nu y.\bar{b}\langle(y,z)\rangle \\
\frac{\text{PAR-L}}{\nu x.\bar{a}\langle(x,z)\rangle \mid \nu y.\bar{b}\langle(y,z)\rangle \xrightarrow{\bar{a}(\lambda)} \nu x.\{\langle(x,z)/\lambda\rangle \mid 0\} \mid \nu y.\bar{b}\langle(y,z)\rangle} \quad \frac{\text{PAR-L}}{a(x_1).P \xrightarrow{a(x_2)} \text{id} \mid P\{\langle(x_2)/x_1\rangle\}} \quad \text{INP} \\
\frac{\text{PAR-L}}{\nu x.\bar{a}\langle(x,z)\rangle \mid \nu y.\bar{b}\langle(y,z)\rangle \xrightarrow{\bar{a}(\lambda)} \nu x.\{\langle(x_2)/\lambda\rangle \mid 0\} \mid \nu y.\bar{b}\langle(y,z)\rangle} \quad \frac{\text{PAR-L}}{a(x_1).P \mid b(x_2).Q \xrightarrow{a(x_2)} \text{id} \mid P\{\langle(x_2)/x_1\rangle\} \mid b(x_2).Q} \quad \text{PAR-L} \\
\frac{\text{CLOSE-L}}{(\nu x.\bar{a}\langle(x,z)\rangle \mid \nu y.\bar{b}\langle(y,z)\rangle) \mid (a(x_1).P \mid b(x_2).Q) \xrightarrow{\tau} \nu x.\{\langle(x_2)/\lambda\rangle \mid 0\} \mid \nu y.\bar{b}\langle(y,z)\rangle \mid (P\{\langle(x_2)/x_1\rangle\} \mid b(x_2).Q)} \quad \text{CLOSE-L} \\
\frac{\text{EXT.}}{\nu z.((\nu x.\bar{a}\langle(x,z)\rangle \mid \nu y.\bar{b}\langle(y,z)\rangle) \mid (a(x_1).P \mid b(x_2).Q)) \xrightarrow{\tau} \nu z.\nu x.\{\langle(x_2)/\lambda\rangle \mid 0\} \mid \nu y.\bar{b}\langle(y,z)\rangle \mid (P\{\langle(x_2)/x_1\rangle\} \mid b(x_2).Q))} \quad \text{EXT.} \\
\frac{\text{ALIAS-FREE}}{\text{id} \mid \nu z.((\nu x.\bar{a}\langle(x,z)\rangle \mid \nu y.\bar{b}\langle(y,z)\rangle) \mid (a(x_1).P \mid b(x_2).Q)) \xrightarrow{\tau} \nu z.\nu x.\{\langle(x_2)/x_1\rangle\} \mid (P\{\langle(x_2)/x_1\rangle\} \mid b(x_2).Q))} \quad \text{ALIAS-FREE} \\
\frac{\text{PAR-R}}{x \# a(x_1).P \mid b(x_2).Q \quad \epsilon \# \nu x.\bar{a}\langle(x,z)\rangle \mid \nu y.\bar{b}\langle(y,z)\rangle, x} \quad (*) \\
\frac{\text{PAR-R}}{z, x \# \text{ran}(\text{id}) = \emptyset \quad \text{fa}(\tau) = \emptyset \subseteq \text{dom}(\text{id}) = \emptyset} \quad (**) \\
\frac{\text{OUT}}{\bar{b}\langle(y,z)\rangle \xrightarrow{\bar{b}(\lambda)} \{\langle(y,z)/\lambda\rangle \mid 0\}} \quad y \# \text{fv}(\bar{b}(\lambda)) \\
\frac{\text{EXTRUDE}}{\nu y.\bar{b}\langle(y,z)\rangle \xrightarrow{\bar{b}(\lambda)} \nu y.\{\langle(y,z)/\lambda\rangle \mid 0\}} \quad y \# 0 \\
\frac{\text{PAR-R}}{0 \mid \nu y.\bar{b}\langle(y,z)\rangle \xrightarrow{\bar{b}(\lambda)} \nu y.\{\langle(y,z)/\lambda\rangle \mid 0\} \mid 0} \quad \frac{\text{PAR-R}}{b(x_2).Q \xrightarrow{b(y_2)} \text{id} \mid Q\{\langle(y_2)/x_2\rangle\}} \quad \text{INP} \\
\frac{\text{PAR-R}}{0 \mid \nu y.\bar{b}\langle(y,z)\rangle \xrightarrow{\bar{b}(\lambda)} \nu y.\{\langle(y,z)/\lambda\rangle \mid 0\} \mid 0} \quad \frac{\text{PAR-R}}{P\{\langle(x_2)/x_1\rangle\} \mid b(x_2).Q \xrightarrow{b(y_2)} \text{id} \mid P\{\langle(x_2)/x_1\rangle\} \mid Q\{\langle(y_2)/x_2\rangle\}} \quad \text{PAR-R} \\
\frac{\text{CLOSE-L}}{(0 \mid \nu y.\bar{b}\langle(y,z)\rangle) \mid (P\{\langle(x_2)/x_1\rangle\} \mid b(x_2).Q)) \xrightarrow{\tau} \nu y.\{\langle(y,z)/\lambda\rangle \mid 0\} \mid (P\{\langle(x_2)/x_1\rangle\} \mid Q\{\langle(y_2)/x_2\rangle\})} \quad \text{CLOSE-L} \\
\frac{\text{ALIAS-FREE}}{\text{id} \mid (0 \mid \nu y.\bar{b}\langle(y,z)\rangle) \mid (P\{\langle(x_2)/x_1\rangle\} \mid b(x_2).Q)) \xrightarrow{\tau} \nu y.\{\langle(y,z)/x_1\rangle \mid 0\} \mid (P\{\langle(x_2)/x_1\rangle\} \mid Q\{\langle(y_2)/x_2\rangle\})} \quad \text{ALIAS-FREE} \\
\frac{\text{RES.}}{\nu z.\nu x.\{\langle(x_2)/x_1\rangle\} \mid (P\{\langle(x_2)/x_1\rangle\} \mid b(x_2).Q)) \xrightarrow{\tau} \nu z.\nu x.\nu y.\{\langle(y_2)/x_2\rangle\} \mid (P\{\langle(x_2)/x_1\rangle\} \mid Q\{\langle(y_2)/x_2\rangle\})} \quad \text{RES.} \\
\frac{\text{PAR-R}}{y \# P\{\langle(x_2)/x_1\rangle\} \mid b(x_2).Q \quad \epsilon \# (0 \mid \nu y.\bar{b}\langle(y,z)\rangle), y} \quad (*) \\
\frac{\text{PAR-R}}{y \# \text{ran}(\text{id}) = \emptyset \quad \text{fa}(\tau) = \emptyset \subseteq \text{dom}(\text{id}) = \emptyset} \quad (**)
\end{array}$$

Figure 6 Derivation of the first execution sequence for P_τ .

B Proof outline: event determinism, decomposition, & composition

We prove each condition *event determinism*, *diamond property 1* and *diamond property 2* separately. For event determinism, we first establish a slightly stronger lemma for processes only (Lemma 14), before covering extended processes. The two diamond properties are more involved. For both, we make use of a function (Def. 16) that, for each location prefix, “localises” in a process the corresponding “component”. This is used to pick out components of a process that are active or inactive in a transition. The function also calculates the variables that are to be extruded along the path to the component to be picked out. The decomposition lemmas (Lemmas 18 and 20) use the above function, to pull apart the derivation tree of transitions to get to the exact part of the tree that concerns the component(s) independent from the component(s) of another transition. In both diamond properties, we have two independent transitions, and hence by decomposition, we can identify the common parts of the derivation of both transitions, and the parts of the derivation where the transitions differ. We then appeal to composition lemmas (Lemmas 22 and 23), that construct transitions where we swap the beginning and end of derivations, thereby completing the missing face of each diamond, modulo some permutations of name binders (enabled by equivariance, Sect. 4.3) and substitutions.

We first establish event determinism for processes only, below. The permutation on variables is used to cope with the EXTRUDE rules that possibly applies α -equivalence to rename bound variables.

► **Lemma 14** (determinism for processes). *Assuming ρ is a permutation (bijective operator) on variables such that $\text{dom}(\rho) \# \pi$ we have, if $P \equiv_{\alpha} Q\rho$ and $P \xrightarrow{\pi}_u B$ and $Q \xrightarrow{\pi'}_u C$ then:*

- *if $\pi = \overline{M}(\lambda)$, then $\pi' = \overline{K}(\lambda')$ and $M =_E K$ and, furthermore, given that we have $C = \nu \vec{y}. (\{N/\lambda'\} \mid Q')$ we have also that $B \equiv_{\alpha} (\nu \vec{y}. (\{N/\lambda\} \mid Q'))\rho$;¹*
- *if $\pi = \pi'$, we have $B \equiv_{\alpha} C\rho$.*

Then we extend the above lemma to extended processes, from which event determinism follows by taking the permutation to be the identity.

► **Lemma 15** (determinism for extend processes). *Assuming ρ is a permutation on variables such that $\text{dom}(\rho) \# \pi$, we have, if $A_0 \equiv A_1\rho$ and $A_0 \xrightarrow{\pi}_u B_0$ and $A_1 \xrightarrow{\pi}_u B_1$ then $B_0 \equiv B_1\rho$.*

The proofs of the diamond properties rely on the following function selecting the component of a process corresponding to a location prefix.

► **Definition 16** (components). *Writing **Proc** the set of processes and **Vars** the set of variables, we define inductively a partial function $\text{Comp} : \{0, 1\}^* \rightarrow (\mathbf{Vars}^* \times \mathbf{Proc}) \rightarrow (\mathbf{Vars}^* \times \mathbf{Proc})$ such that $\text{Comp}(\epsilon)(\vec{y}, P) = (\vec{y}, P)$ and if $\mathbf{s} \neq \epsilon$ then it is defined as follows:*

$$\begin{aligned} \text{Comp}(0\mathbf{s})(\vec{y}, P_0 \mid P_1) &= \text{Comp}(\mathbf{s})(\vec{y}, P_0) & \text{Comp}(\mathbf{s})(\vec{y}, \nu x.P) &= \text{Comp}(\mathbf{s})(\vec{y}x, P) \\ \text{Comp}(1\mathbf{s})(\vec{y}, P_0 \mid P_1) &= \text{Comp}(\mathbf{s})(\vec{y}, P_1) & \text{Comp}(\mathbf{s})(\vec{y}, !P) &= \text{Comp}(\mathbf{s})(\vec{y}, P \mid !P) \end{aligned}$$

¹ This means outputs may only differ in the choice of alias (λ v.s. λ') or in that an equivalent recipe (M v.s. K) may be used, and, furthermore, we are free to rename the alias. This clause is a trick used to determine whether the CLOSE-L or CLOSE-R rule applied in an interaction, by looking at the output action and without being required to record additional information in the interaction event.

The decomposition lemmas below select the part of a derivation tree that concerns a single component. We first establish decomposition for simple prefixes consisting of 0 or 1 . In what follows, we write $\neg : \{0, 1\} \rightarrow \{0, 1\}$ for Boolean negation.

- **Lemma 17** (decomposing prefixed transitions). *For all $c \in \{0, 1\}$, if $P \xrightarrow[cu]{\pi} \nu \vec{z}.(\theta \mid Q)$, then:*
- $Comp(c)(\epsilon, P) = (\vec{y}, P')$ and $\vec{z} = \vec{y}, \vec{x}$ and $P' \xrightarrow[u]{\pi} \nu \vec{x}.(\theta \mid Q')$ and $Comp(c)(\epsilon, Q) = (\epsilon, Q')$.
 - In addition, $Comp(\neg c)(\epsilon, P) = (\vec{y}, R)$ and $Comp(\neg c)(\epsilon, Q) = (\epsilon, R)$ and $\vec{x} \# R$.²

We then appeal to the fact that $Comp(s)(Comp(s')(h, P)) = Comp(s's)(h, P)$ and make use of Lemma 17 repeatedly to generalise decomposition to an arbitrary string.

- **Lemma 18** (decomposing process transitions). *For all $s \in \{0, 1\}^*$ and $P \xrightarrow[su]{\pi} \nu \vec{z}.(\theta \mid Q)$ then we have the following:*
- $Comp(s)(\epsilon, P) = (\vec{y}, P')$ and $\vec{z} = \vec{y}, \vec{x}$ and $P' \xrightarrow[u]{\pi} \nu \vec{x}.(\theta \mid Q')$ and $Comp(s)(\epsilon, Q) = (\epsilon, Q')$;
 - for any $s', s'' \in \{0, 1\}^*$, and $c \in \{0, 1\}$, if $s = s'cs''$ then $Comp(s'-c)(\epsilon, P) = (\vec{w}, R)$ and $Comp(s'-c)(\epsilon, Q) = (\epsilon, R)$ and $Comp(s'c)(\epsilon, P) = (\vec{w}, S)$ and $Comp(s'')(\epsilon, S) = (\vec{v}, P')$, and also $\vec{v}, \vec{x} \# R$.

A richer decomposition lemma is needed for τ -transitions, so that we can identify the two components – for the input and output transition involved in the communication – that are both structurally independent of another transition.

- **Lemma 19** (interaction decomposition). *If $P \xrightarrow[(0\ell_0, 1\ell_1)]{\tau} \nu \vec{z}.(\theta \mid Q)$ then we have $Comp(0)(\epsilon, P) = (\vec{y}, P_0)$ and $Comp(1)(\epsilon, P) = (\vec{y}, P_1)$ and $Comp(0)(\epsilon, Q) = (\epsilon, Q_0)$ and $Comp(1)(\epsilon, Q) = (\epsilon, Q_1)$ and $\vec{z} = \vec{y}, \vec{x}_0, \vec{x}_1$ and $\vec{x}_0 \# Q_1$ and $\vec{x}_1 \# Q_0$ and $\vec{x}_0 \# \vec{x}_1$ and one of the following hold:*

- $P_0 \xrightarrow[\ell_0]{\overline{M}(\lambda)} \nu \vec{x}_0.(\{N/\lambda\} \mid Q_0)$ and $P_1 \xrightarrow[\ell_1]{MN} \nu \vec{x}_1.(\text{id} \mid Q_1)$.
- $P_0 \xrightarrow[\ell_0]{MN} \nu \vec{x}_0.(\text{id} \mid Q_0)$ and $P_1 \xrightarrow[\ell_1]{\overline{M}(\lambda)} \nu \vec{x}_1.(\{N/\lambda\} \mid Q_1)$.

Again appealing to that fact that $Comp(s)(Comp(s')(h, P)) = Comp(s's)(h, P)$ we can generalise decomposition of interactions to an arbitrary prefix string.

- **Lemma 20** (full decomposition of interactions). *For all $s, s_0, s_1 \in \{0, 1\}^*$ such that we have $P \xrightarrow[s(0s_0\ell_0, 1s_1\ell_1)]{\tau} \nu \vec{z}.(\theta \mid Q)$, the following hold:*
- we have $Comp(s0s_0)(\epsilon, P) = (\vec{y}\vec{z}_0, P_0)$ and $Comp(s1s_1)(\epsilon, P) = (\vec{y}\vec{z}_1, P_1)$ and, also we have $Comp(s0s_0)(\epsilon, Q) = (\epsilon, Q_0)$ and $Comp(s1s_1)(\epsilon, Q) = (\epsilon, Q_1)$ and $\vec{z} = \vec{y}, \vec{z}_0, \vec{x}_0, \vec{z}_1, \vec{x}_1$ and $\vec{z}_0, \vec{x}_0 \# Q_1$ and $\vec{z}_1, \vec{x}_1 \# Q_0$ and $\vec{z}_0, \vec{x}_0 \# \vec{z}_1, \vec{x}_1$ and one of the following hold:
 - $P_0 \xrightarrow[\ell_0]{\overline{M}(\lambda)} \nu \vec{x}_0.(\{N/\lambda\} \mid Q_0)$ and $P_1 \xrightarrow[\ell_1]{MN} \nu \vec{x}_1.(\text{id} \mid Q_1)$,
 - $P_0 \xrightarrow[\ell_0]{MN} \nu \vec{x}_0.(\text{id} \mid Q_0)$ and $P_1 \xrightarrow[\ell_1]{\overline{M}(\lambda)} \nu \vec{x}_1.(\{N/\lambda\} \mid Q_1)$;
 - for any $s', s'' \in \{0, 1\}^*$, and $c \in \{0, 1\}$ and $s' \neq s$, we have the following: if $s0s_0 = s'cs''$ or $s1s_1 = s'cs''$ then $Comp(s'-c)(\epsilon, P) = (\vec{w}, R)$ and $Comp(s'-c)(\epsilon, Q) = (\epsilon, R)$ and also $Comp(s'c)(\epsilon, P) = (\vec{w}, S)$ and $Comp(s'')(\epsilon, S) = (\vec{v}, P')$, and we have $\vec{v}, \vec{x}_0, \vec{x}_1 \# R$.

² This states that the locations independent from P' are unchanged by the transition stemming from P' , except that the common name binders will be extruded. Generalisations of this statement carry through to the other decomposition lemmas.

Composition lemmas are required to remember the part of the derivation tree picked out by the decomposition lemmas, when constructing a new transition. As for decomposition, we first establish composition for a single-character prefix: **0** or **1**.

► **Lemma 21** (composing in one-step). *For any $s \in \{0, 1\}$, if $\text{Comp}(s)(P) = (\vec{y}, P')$ and we have $P' \xrightarrow[u]{\pi} \nu \vec{z}.(\sigma \mid Q')$ and, furthermore, $\text{Comp}(\neg s)(P) = (\vec{y}, R)$ and $\vec{z} \# R$, then, for some Q , we have $P \xrightarrow[su]{\pi} \nu \vec{y}, \vec{z}.(\sigma \mid Q)$ and $\text{Comp}(s)(\epsilon, Q) = (\epsilon, Q')$.*

As for decomposition, we extend composition to any prefix.

► **Lemma 22** (composing transitions). *Assume $s \in \{0, 1\}^*$, and $\text{Comp}(s)(P) = (\vec{y}, P')$ and we have $P' \xrightarrow[u]{\pi} \nu \vec{z}.(\sigma \mid Q')$, and, furthermore, for any $s', s'' \in \{0, 1\}^*$, and $c \in \{0, 1\}$ such that $s = s'cs''$, we have $\text{Comp}(s'-c)(\epsilon, P) = (\vec{w}, R)$ and $\text{Comp}(s'c)(\epsilon, P) = (\vec{w}, S)$ and $\text{Comp}(s'')(c, S) = (\vec{v}, P')$, and also $\vec{v}, \vec{x} \# R$. Given these assumptions, we have that, for some Q , we have $P \xrightarrow[su]{\pi} \nu \vec{y}, \vec{z}.(\sigma \mid Q)$ and $\text{Comp}(s)(\epsilon, Q) = (\epsilon, Q')$.*

Interactions can also be composed.

► **Lemma 23** (composing interactions). *Assume $s, s_0, s_1 \in \{0, 1\}$, are such that we have $\text{Comp}(s0)(\epsilon, P) = (\vec{y}, Q_0)$ and $\text{Comp}(s1)(\epsilon, P) = (\vec{y}, Q_1)$ and $\text{Comp}(s_0)(\epsilon, Q_0) = (\vec{x}_0, P_0)$ and $\text{Comp}(s_1)(\epsilon, Q_1) = (\vec{x}_1, P_1)$ and either of the following hold:*

- $P_0 \xrightarrow[\ell_0]{\overline{M}(\lambda)} \nu \vec{z}.(\{N/\lambda\} \mid P'_0)$ and $P_1 \xrightarrow[\ell_1]{MN} \nu \vec{w}.(\text{id} \mid P'_1)$,
- $P_0 \xrightarrow[\ell_0]{MN} \nu \vec{z}.(\text{id} \mid P'_0)$ and $P_1 \xrightarrow[\ell_1]{\overline{M}(\lambda)} \nu \vec{w}.(\{N/\lambda\} \mid P'_1)$;

and also assume we have, for any $s', s'' \in \{0, 1\}^*$, and $c \in \{0, 1\}$, such that $s' \neq s$ and either $s0s_0 = s'cs''$ or $s1s_1 = s'cs''$, we have that $\text{Comp}(s'-c)(\epsilon, P) = (\vec{w}, R)$ and also $\text{Comp}(s'c)(\epsilon, P) = (\vec{w}, S)$ and $\text{Comp}(s'')(c, S) = (\vec{v}, P')$, and $\vec{v}, \vec{z}, \vec{w} \# R$. Under those assumptions, for some P' , we have the transition $P \xrightarrow[su]{\pi} \nu \vec{y}, \vec{x}_0, \vec{z}, \vec{x}_1, \vec{w}.(\sigma \mid P')$ and also we have $\text{Comp}(s0s_0)(\epsilon, P) = (\epsilon, P'_0)$ and $\text{Comp}(s1s_1)(\epsilon, P) = (\epsilon, P'_1)$.

Using the decomposition and composition lemmas, we can now construct the transitions required to complete the two diamond properties.

► **Lemma 24** (diamond property 1). *If $(\pi_0, u_0) I (\pi_1, u_1)$, $A \xrightarrow[u_0]{\pi_0} B_0$ and $A \xrightarrow[u_1]{\pi_1} B_1$, then $\exists C_0, C_1$ s.t. $B_0 \xrightarrow[u_1]{\pi_1} C_0$ and $B_1 \xrightarrow[u_0]{\pi_0} C_1$ and $C_0 \equiv C_1$.*

In both diamond properties, there are several cases depending on what combination of τ and output labelled events we are considering to be independent. Below we present the top-level breakdown of the case analysis, which applies to both diamond properties. We also provide the details of one of the most interesting cases, where two independent output transitions occur.

► **Lemma 25** (diamond property 2). *If $(\pi_0, u_0) I (\pi_1, u_1)$, $A \xrightarrow[u_0]{\pi_0} B_0$ and $B_0 \xrightarrow[u_1]{\pi_1} C_0$, then $\exists B_1, C_1$ s.t. $A \xrightarrow[u_1]{\pi_1} B_1$ and $B_1 \xrightarrow[u_0]{\pi_0} C_1$ and $C_0 \equiv C_1$.*

Proof. Assume we have $(\pi_0, u_0) I (\pi_1, u_1)$, and the two transitions $A \xrightarrow[u_0]{\pi_0} B_0$ and $B_0 \xrightarrow[u_1]{\pi_1} C_0$.

For two transitions from the same state we have $(\pi_0, u_0) I (\pi_1, u_1)$ iff for all $\ell_0 \in \mathcal{Loc}(u_0)$ and for all $\ell_1 \in \mathcal{Loc}(u_1)$, we have $\ell_0 I_{\ell_1}$ (i.e., there is no structural causality), and, furthermore, if $\pi_0 = \overline{M}_0(\alpha_0)$, then $\alpha_0 \# \pi_1$ (i.e., there is no link causality). The structural independence ensures that, without loss of generality, we have one of the following.

- Both location labels have a largest common prefix $s \in \{0, 1\}^*$ and hence are of the form $s0u'_0 = u_0$ and $s1u'_1 = u_1$.
- At least one transition is labelled with a τ action (without loss of generality let $\pi_0 = \tau$), and so $u_0 = s0(\ell_0, 1\ell_1)$ and there is a string $s_1 \in \{0, 1\}^*$, characters $c, d \in \{0, 1\}$ and prefix location ℓ' such that $s_0ds_1cu'_1 = u_1$ and $s_1\text{-}c\ell' = \ell_d$. That is, one transition is an interaction, and the other transition is entirely located within one of the locations from which either interacting input or the interacting output emanated.
- Both are τ transitions with a common prefix $s \in \{0, 1\}^*$ such that $u_0 = s(0\ell_0^0, 1\ell_0^1)$ and $u_1 = s(0\ell_1^0, 1\ell_1^1)$ and there are strings $s_1 \in \{0, 1\}^*$ such that $\ell_0^0 = s_00\ell_0^0$ and $\ell_1^0 = s_01\ell_1^0$ and $\ell_0^1 = s_10\ell_0^1$ and $\ell_1^1 = s_11\ell_1^1$. That is, we have two interactions, where the interaction occurs in the same location (even though the inputs and outputs involved are independent).

We break down further the first case above, where the two independent transitions have a common prefix. Due to differences between ALIAS-OUT or and ALIAS-FREE, we should consider separately the cases where the first transition is an output transition or a free transition. We consider only the most interesting case, where we appeal to the absence of link causality, which is only relevant when $\pi_0 = \overline{M_0}(\alpha_0)$ is an output transition. That case itself breaks down into two cases, where π_1 is either another output transition or a free transition. We provide only the case when π_1 is an output transition such that $\pi_1 = \overline{M_1}(\alpha_1)$ below.

Without loss of generality (0 and 1 can be reversed without changing the proof), assume there exists s such that $u_0 = \ell_0 = s0s_0[t_0]$ and $u_1 = \ell_1 = s1s_1[t_1]$.

Thus, by the RES rule, repeatedly, we have $A = \nu\vec{x}.\langle\sigma \mid P\rangle$ and $\alpha_0 = s0s_0\lambda_0$ and $\vec{x} \# M_0$ and $B_0 = \nu\vec{x}, \vec{y}_0.\langle\sigma \circ \{N_0/s_0s_0\lambda_0\} \mid Q_0\rangle$, and, also by the ALIAS-OUT rule we have the following.

$$\frac{P \xrightarrow[\text{s0s}_0[t_0]]{\overline{M_0}\sigma(\lambda_0)} \nu\vec{y}_0.\langle\{N_0/\lambda_0\} \mid Q_0\rangle \quad \vec{y} \# \text{ran}(\sigma) \quad \text{fa}(M_0) \subseteq \text{dom}(\sigma) \quad \text{s0s}_0\lambda_0 \# \text{dom}(\sigma)}{\frac{\sigma \mid P \xrightarrow[\text{s0s}_0[t_0]]{\overline{M_0}(s_0s_0\lambda_0)} \nu\vec{y}_0.\langle\sigma \circ \{N_0/s_0s_0\lambda_0\} \mid Q_0\rangle}{\nu\vec{x}.\langle\sigma \mid P\rangle \xrightarrow[\text{s0s}_0[t_0]]{\overline{M_0}(s_0s_0\lambda_0)} \nu\vec{x}, \vec{y}_0.\langle\sigma \circ \{N_0/s_0s_0\lambda_0\} \mid Q_0\rangle}}$$

Now, since we have $P \xrightarrow[\text{s0s}_0[t_0]]{\overline{M_0}\sigma(\lambda_0)} \nu\vec{y}_0.\langle\{N_0/\lambda_0\} \mid Q_0\rangle$, by Lemma 18, we have the following:

- $P_0 \xrightarrow[\text{s}_0[t_0]]{\overline{M_0}\sigma(\lambda_0)} \nu\vec{z}_0.\langle\{N_0/\lambda_0\} \mid Q'_0\rangle$ and $\text{Comp}(s_0)(\epsilon, P) = (\vec{y}, P_0)$ and $\vec{y}_0 = \vec{y}, \vec{z}_0$ and $\text{Comp}(s_0)(\epsilon, Q_0) = (\vec{y}, Q'_0)$.
- for any $s', s'' \in \{0, 1\}^*$, and $c \in \{0, 1\}$, if $s_0 = s'cs''$ then $\text{Comp}(s'\text{-}c)(\epsilon, P) = (\vec{w}, R)$ and $\text{Comp}(s'\text{-}c)(\epsilon, Q_0) = (\epsilon, R)$ and $\text{Comp}(s'c)(\epsilon, P) = (\vec{w}, S)$ and $\text{Comp}(s'')(\epsilon, S) = (\vec{v}, P_0)$, and also $\vec{v}, \vec{x} \# R$.

Now since $B_0 = \nu\vec{x}, \vec{y}_0.\langle\sigma \circ \{N_0/s_0s_0\lambda_0\} \mid Q_0\rangle$ and $B_0 \xrightarrow[\ell_1]{\pi_1} C$, by the RES rule repeatedly and the ALIAS-OUT rule, we have the following, where $\vec{x}, \vec{y}_0 \# M_1$ and $\theta_0 = \sigma \circ \{N_0/s_0s_0\lambda_0\}$ and $C_0 = \nu\vec{x}, \vec{y}_0, \vec{z}_1.\langle\theta_0 \circ \{N_1/s_1s_1\lambda_1\} \mid R_0\rangle$.

$$\frac{Q_0 \xrightarrow[\text{s1s}_1[t_1]]{\overline{M_1}\sigma(\lambda_1)} \nu\vec{z}_1.\langle\{N_1/\lambda_1\} \mid R_0\rangle \quad \vec{z}_1 \# \text{ran}(\theta_0) \quad \text{fa}(M_1) \subseteq \text{dom}(\theta_0) \quad \text{s1s}_1\lambda_1 \# \text{dom}(\theta_0)}{\frac{\theta_0 \mid Q_0 \xrightarrow[\text{s1s}_1[t_1]]{\overline{M_1}(s_1s_1\lambda_1)} \nu\vec{z}_1.\langle\theta_0 \circ \{N_1/s_1s_1\lambda_1\} \mid R_0\rangle}{\nu\vec{x}, \vec{y}_0.\langle\theta_0 \mid Q_0\rangle \xrightarrow[\text{s1s}_1[t_1]]{\overline{M_1}(s_1s_1\lambda_1)} \nu\vec{x}, \vec{y}_0, \vec{z}_1.\langle\theta_0 \circ \{N_1/s_1s_1\lambda_1\} \mid R_0\rangle}}$$

Since $Q_0 \xrightarrow[\mathbf{s1s1}[t_1]]{\overline{M_1\sigma}(\lambda_1)} \nu \bar{z}_1.(\{M_1/\lambda_1\} \mid R_0)$, by Lemma 18, we have the following:

- $Comp(\mathbf{s1})(\epsilon, Q_0) = (\mathbf{s1}, Q'_0)$ and $Q'_0 \xrightarrow[\mathbf{s1}[t_1]]{\overline{M_1\sigma}(\lambda_1)} \nu \bar{z}_1.(\{M_1/\lambda_1\} \mid R'_0)$ and $Comp(\mathbf{s1})(\epsilon, R_0) = (\epsilon, R'_0)$.
- for any $s', s'' \in \{0, 1\}^*$, and $c \in \{0, 1\}$, if $\mathbf{s1} = s'cs''$ then $Comp(s'\neg c)(\epsilon, Q_0) = (\bar{w}, R)$ and $Comp(s'\neg c)(\epsilon, R_0) = (\epsilon, R)$ and $Comp(s'c)(\epsilon, Q_0) = (\bar{w}, S)$ and $Comp(s'')(\epsilon, S) = (\bar{v}, Q'_0)$, and also $\bar{v}, \bar{x} \# R$.

From the above we have that $Comp(\mathbf{s1})(\epsilon, P) = (\bar{y}, Q'_0)$ and $Comp(\mathbf{s1})(\epsilon, Q_0) = (\epsilon, Q'_0)$. We also have that, for any $s', s'' \in \{0, 1\}^*$, and $c \in \{0, 1\}$, if $s = s'cs''$ then $Comp(s'\neg c)(\epsilon, P_0) = (\bar{w}, R)$ and $Comp(s'\neg c)(\epsilon, Q_0) = (\epsilon, R)$ and $Comp(s'\neg c)(\epsilon, R_0) = (\epsilon, R)$ and $Comp(s'c)(\epsilon, Q_0) = (\bar{w}, S)$ and $Comp(s'')(\epsilon, S) = (\bar{v}, Q'_0)$, and also $\bar{v}, \bar{x} \# R$.

We now appeal to the absence of *link causality*, so we know that $\mathbf{s0s0}\lambda_0 \# M_1$, and hence $M_1\theta_0 = M_1\sigma$, and so $Q'_0 \xrightarrow[\mathbf{s1}[t_1]]{\overline{M_1\sigma}(\lambda_1)} \nu \bar{z}_1.(\{N_1/\lambda_1\} \mid R'_0)$. Therefore by Lemma 22, we have $P \xrightarrow[\mathbf{s1s1}[t_1]]{\overline{M_1\sigma}(\lambda_1)} \nu \bar{y}, \bar{z}_1.(\{N_1/\lambda_1\} \mid Q_1)$ and $Comp(\mathbf{s1})(\epsilon, Q_1) = (\epsilon, R'_0)$. Since we know $\text{fa}(M_1) \subseteq \text{dom}(\theta_0)$ and $\mathbf{s0s0}\lambda_0 \# M_1$ we know that $\text{fa}(M_1) \subseteq \text{dom}(\sigma)$. Since $\mathbf{s1s1}\lambda_1 \# \text{dom}(\theta_0)$ we have $\mathbf{s1s1}\lambda_1 \# \text{dom}(\sigma)$. Thus, by ALIAS-OUT and RES repeatedly, we have the following.

$$\frac{P \xrightarrow[\mathbf{s1s1}[t_1]]{\overline{M_1\sigma}(\lambda_1)} \nu \bar{y}, \bar{z}_1.(\{N_1/\lambda_1\} \mid Q_1) \quad \text{fa}(M_1) \subseteq \text{dom}(\sigma) \quad \bar{y}, \bar{z}_1 \# \text{ran}(\sigma) \quad \mathbf{s1s1}\lambda_1 \# \text{dom}(\sigma)}{\sigma \mid P \xrightarrow[\mathbf{s1s1}[t_1]]{\overline{M_1\sigma}(\mathbf{s1s1}\lambda_1)} \nu \bar{y}, \bar{z}_1.(\sigma \circ \{N_1/\mathbf{s1s1}\lambda_1\} \mid Q_1)}$$

$$\frac{\sigma \mid P \xrightarrow[\mathbf{s1s1}[t_1]]{\overline{M_1\sigma}(\mathbf{s1s1}\lambda_1)} \nu \bar{y}, \bar{z}_1.(\sigma \circ \{N_1/\mathbf{s1s1}\lambda_1\} \mid Q_1)}{\nu \bar{x}.(\sigma \mid P) \xrightarrow[\mathbf{s1s1}[t_1]]{\overline{M_1\sigma}(\mathbf{s1s1}\lambda_1)} \nu \bar{x}, \bar{y}, \bar{z}_1.(\sigma \circ \{N_1/\mathbf{s1s1}\lambda_1\} \mid Q_1)}$$

Recall that $A = \nu \bar{x}.(\sigma \mid P)$, hence we have the first of our desired transitions.

It remains to show that $\nu \bar{x}, \bar{y}, \bar{z}_1.(\theta_1 \mid Q_1) \xrightarrow[u_0]{\pi_0} C_1$, where $\theta_1 = \sigma \circ \{N_1/\mathbf{s1s1}\lambda_1\}$, and also $C_0 \equiv C_1$. Since $\text{fa}(M_0) \subseteq \text{dom}(\sigma) \subseteq \text{dom}(\theta_1)$, we have $M_0\sigma = M_0\theta_1$, thus $P_0 \xrightarrow[\mathbf{s0}[t_0]]{\overline{M_0\theta_1}(\lambda_0)} \nu \bar{z}_0.(\{N_0/\lambda_0\} \mid Q'_1)$. Therefore, since we know (via Lemma 18) that $Comp(\mathbf{s0})(\epsilon, Q_1) = (\epsilon, P_0)$, by Lemma 22, $Q_1 \xrightarrow[\mathbf{s0}[t_0]]{\overline{M_0\theta_1}(\lambda_0)} \nu \bar{z}_0.(\{N_0/\lambda_0\} \mid R_1)$ and $Comp(\mathbf{s0})(\epsilon, R_1) = (\epsilon, Q'_0)$.

By the RES rule repeatedly and the ALIAS-OUT rule, we have that $\bar{x}, \bar{y}, \bar{z}_1 \# M_0$ and we have the following transition, and so $C_1 = \nu \bar{x}, \bar{y}, \bar{z}_1, \bar{z}_0.(\theta_1 \circ \{N_0/\mathbf{s0s0}\lambda_0\} \mid R_1)$.

$$\frac{Q_1 \xrightarrow[\mathbf{s0s0}[t_0]]{\overline{M_0}(\lambda_0)} \nu \bar{z}_0.(\{N_0/\mathbf{s0s0}\lambda_0\} \mid R_1) \quad \text{fa}(M_0) \subseteq \text{dom}(\theta_1) \quad \bar{z}_0 \# \text{ran}(\theta_1) \quad \mathbf{s0s0}\lambda_0 \# \text{dom}(\theta_1)}{\theta_1 \mid Q_1 \xrightarrow[\mathbf{s0s0}[t_0]]{\overline{M_0}(\mathbf{s0s0}\lambda_0)} \nu \bar{z}_0.(\sigma \circ \{N_1/\mathbf{s1s1}\lambda_1\} \circ \{N_0/\mathbf{s0s0}\lambda_0\} \mid R_1)}$$

$$\frac{\theta_1 \mid Q_1 \xrightarrow[\mathbf{s0s0}[t_0]]{\overline{M_0}(\mathbf{s0s0}\lambda_0)} \nu \bar{z}_0.(\sigma \circ \{N_1/\mathbf{s1s1}\lambda_1\} \circ \{N_0/\mathbf{s0s0}\lambda_0\} \mid R_1)}{\nu \bar{x}, \bar{y}, \bar{z}_1.(\theta_1 \mid Q_1) \xrightarrow[\mathbf{s0s0}[t_0]]{\overline{M_0}(\mathbf{s0s0}\lambda_0)} \nu \bar{x}, \bar{y}, \bar{z}_1, \bar{z}_0.(\sigma \circ \{N_1/\mathbf{s1s1}\lambda_1\} \circ \{N_0/\mathbf{s0s0}\lambda_0\} \mid R_1)}$$

Now since for all $s', s'' \in \{0, 1\}$ and $c, d \in \{0, 1\}$ such that $s'cs'' = sd$ we have $Comp(s'\neg c)(\epsilon, R_0) = Comp(s'\neg c)(\epsilon, R_1)$ (via Lemma 18 and the above), clearly it is the case that $R_0 = R_1$. Furthermore, we have the following, as required.

$$C_0 = \nu \bar{x}, \bar{y}, \bar{z}_0, \bar{z}_1.(\sigma \circ \{N_0/\mathbf{s0s0}\lambda_0\} \circ \{N_1/\mathbf{s1s1}\lambda_1\} \mid R_0)$$

$$\equiv \nu \bar{x}, \bar{y}, \bar{z}_1, \bar{z}_0.(\sigma \circ \{N_1/\mathbf{s1s1}\lambda_1\} \circ \{N_0/\mathbf{s0s0}\lambda_0\} \mid R_1) = C_1$$

Other cases follow a similar pattern of applying to decomposition and composition. ◀