Under consideration for publication in Math. Struct. in Comp. Science

Proof-relevant π-calculus

Roly Perera[1] [†] James Cheney[2] [‡]

[1] *University of Glasgow*
[2] *University of Edinburgh*

Received 18 April 2016

We present a formalisation in Agda of the theory of concurrent transitions, residuation, and causal equivalence of traces for the π-calculus. Our formalisation employs de Bruijn indices and dependently-typed syntax, and aligns the "proved transitions" proposed by Boudol and Castellani in the context of CCS with the proof terms naturally present in Agda's representation of the labelled transition relation. Our main contributions are proofs of the "diamond lemma" for the residuals of concurrent transitions and a formal definition of equivalence of traces up to permutation of transitions.

In the π-calculus transitions represent propagating binders whenever their actions involve bound names. To accommodate these cases, we require a more general diamond lemma where the target states of equivalent traces are no longer identical, but are related by a *braiding* that rewires the bound and free names to reflect the particular interleaving of events involving binders. Our approach may be useful for modelling concurrency in other languages where transitions carry metadata sensitive to particular interleavings, such as dynamically allocated memory addresses.

1. Introduction

The π-calculus [Milner 1999; Milner et al. 1992] is an expressive model of concurrent and mobile processes. It has been investigated extensively and many variants, extensions and refinements proposed, including the asynchronous, polyadic, and applied π-calculus [Sangiorgi and Walker 2001]. The π-calculus has also attracted considerable attention from the logical frameworks and meta-languages community, and formalisations of its syntax and semantics have been developed in most of the extant mechanised metatheory systems, including HOL [Melham 1994; Aït Mohamed 1995], Coq [Hirschkoff 1997; Despeyroux 2000; Honsell et al. 2001], Isabelle/HOL [Röckl et al. 2001; Gay 2001], Nominal Isabelle [Bengtson and Parrow 2009], Abella [Baelde et al. 2014], CLF [Cervesato et al. 2002], and Agda [Orchard and Yoshida 2016]. Indeed, some early formalisations motivated or led to important developments in mechanised metatheory, such as the Theory of Contexts, or CLF's support for monadic encapsulation of concurrent executions.

Prior formalisations have typically considered the syntax, semantics and bisimulation

[†] Supported by UK EPSRC grant EP/K034413/1 and US AFOSR grant FA8655-13-1-3006.
[‡] Supported by a Royal Society University Research Fellowship.

theory of the π-calculus. One interesting aspect of the π-calculus that has not been formally investigated, and remains to some extent ill-understood informally, is its theory of *causal equivalence*. Two transitions t, t' that can be taken from a process term P are said to be *concurrent*, written $t \smile t'$, if they can be performed "in either order" — that is, if after performing t, there is a natural way to transform the other transition t' so that its effect is performed on the result of t, and vice versa. The transformed version of the transition is said to be the *residual* of t' after t, written t'/t. The key property of this operation, called the "diamond lemma" [Lévy 1980], is that the two residuals t/t' and t'/t result in the same process. Finally, permutation of concurrent transitions induces a *causal equivalence* relation on pairs of traces. This relation is the standard notion of permutation-equivalence from the theory of traces over concurrent alphabets [Mazurkiewicz 1987].

Our interest in this area stems from previous work that established a connection between backward slicing and reversibility [Perera et al. 2012]. We wish to adapt the approach of the earlier work to concurrent languages, which requires being able to represent, manipulate, and reason about concurrent executions safely: that is, respecting well-formedness and causality. The π-calculus is a natural starting point for this study.

In classical treatments of concurrency and residuation, starting with Lévy [1980], a transition is usually considered to be a triple (e, t, e') where e and e' are the source and target terms of the transition and t is some information about the step performed. Boudol and Castellani [1989] introduced the *proved transitions* approach for CCS in which the labels of transitions are enriched with an approximation of the derivation tree which proves that a particular triple is in the transition relation. Boreale and Sangiorgi [1998] and Degano and Priami [1999] developed theories of causal equivalence for the π-calculus, building indirectly on the proved transition approach; Danos and Krivine [2004] and Cristescu et al. [2013] developed notions of causality in the context of reversible CCS and π-calculus respectively.

None of the above treatments has been mechanised, although the theory of residuals for the λ-calculus was formalised in Coq by Huet [1994] and in Abella by Accattoli [2012]. In this paper, we report on a formalisation of concurrency, residuation and causal equivalence for the π-calculus carried out in the dependently-typed programming language Agda [Norell 2009]. Our approach is inspired by the proved transitions method of Boudol and Castellani. However, by taking a "Church-style" view of the labelled transition semantics and treating transitions as proof terms, rather than triples (e, t, e'), we avoid the need for an auxiliary notion of "proved transition". Agda's dependent typing allows us to define the concurrency relation on (compatibly-typed) transition proofs, and residuation as a total function taking two transitions along with a proof that the transitions are concurrent. Our formalisation employs de Bruijn indices [de Bruijn 1972], an approach with well-known strengths and weaknesses compared, for example, to higher-order or nominal abstract syntax techniques employed in existing formalisations.

Our definition of concurrency is not the only plausible one for the π-calculus. Indeed, there appears to be little consensus regarding the characteristics of a canonical definition. For example, Cristescu et al. [2013] write "[in] the absence of an indisputable definition of permutation equivalence for [labelled transition system] semantics of the π-calculus it is hard to assert the correctness of one definition over another." We do, however, show that our definition of concurrency is sound by proving the diamond property; to the best of our knowledge, ours is the first mechanised version of this result for any process calculus.

However, one key observation that emerges in our development is that requiring residuals of concurrent transitions to reach exactly the same state is too restrictive. When the action of a transition involves a bound name, the transition represents a propagating binder. In such cases equivalent traces no longer have identical target states, but rather states which are equal up to a *braiding* that rewires the bound and free names to reflect the different order of events in the two traces. Although typically unobservable to a program, such interleaving-sensitive information may be important for other purposes, such as memory locations in a debugger, or transaction ids in a financial application. In these situations being able to robustly translate between the target states of different interleavings may be important. Our development may therefore be a useful case study for formalising concurrency in other settings where transition labels carry interleaving-sensitive metadata. It may also be an illuminating use case for further developments in dependently-typed programming, such as quotient types and higher inductive types.

This is an extended version of a paper presented at the *Logical Frameworks and Meta-Languages: Theory and Practice* workshop [Perera and Cheney 2015]. This version extends the earlier work with graphical proof-sketches for various lemmas, a more detailed comparison of related formalisation efforts, extensive examples and discussion regarding the generalised diamond property, and a formalisation of composite braids that was omitted from the earlier work.

The paper is organised as follows. § 2 describes our variant of the (synchronous) π-calculus, including syntax, renamings, and transitions. § 3 defines concurrency and residuation for transitions, and discusses the diamond lemma and the notion of "cofinal" transitions. § 4 presents our definition of causal equivalence. § 5 discusses related work in more detail and § 6 concludes and discusses prospects for future work. Appendix A summarises the Agda module structure; the source code can be found at `https://github.com/rolyp/proof-relevant-pi`, release `0.2`.

2. Synchronous π-calculus

We present our formalisation in the setting of a first-order, synchronous, monadic π-calculus with recursion and internal choice, using a labelled transition semantics. The syntax of the calculus is conventional (using de Bruijn indices) and is given below.

Name	x, y, z	::=	$0 \mid 1 \mid \cdots$		Process	P, Q, R, S	::=	$\mathbf{0}$	inactive
Action	a	::=	$\underline{x}$	input				$\underline{x}.P$	input
			$\overline{x}\langle y \rangle$	output				$\overline{x}\langle y \rangle.P$	output
			$\overline{x}$	bound output				$P + Q$	choice
			τ	silent				$P \mid Q$	parallel
								νP	restriction
								$!P$	replication

Names are ranged over by x, y and z. An input action is written $\underline{x}$. Output actions are written $\overline{x}\langle y \rangle$ if y is in scope and $\overline{x}$ if the action represents the output of a name whose scope is extruding, in which case we say the action is a *bound* output. Bound outputs do not appear in user code but arise during execution.

The formal development in Agda uses de Bruijn indices, and we give definitions and state properties in terms of this notation, but we will sometimes illustrate their meaning in terms of conventional π-calculus notation. For example, the conventional π-calculus term $(\nu x)\ x(z).\overline{y}\langle z\rangle.\mathbf{0} \mid \overline{x}\langle c\rangle.\mathbf{0}$ would be represented using de Bruijn indices as $\nu(\underline{0}.\overline{n+1}\langle 0\rangle.\mathbf{0} \mid \overline{0}\langle m+1\rangle.\mathbf{0})$, provided that y and c are associated with indices n and m. Here, the first 0 represents the bound variable x, the second 0 the bound variable z, and the third refers to x again. Note that the symbol $\mathbf{0}$ denotes the inactive process term, not a de Bruijn index.

Let Γ and Δ range over *contexts*, which in an untyped setting are simply natural numbers. A membership witness $x \in \Gamma$ is a proof that $x < \Gamma$. A context Γ *closes* P if $x \in \Gamma$ for every free variable x of P. We denote by $\mathsf{Proc}\ \Gamma$ the set of processes closed by Γ, as defined below. We write $\Gamma \vdash P$ to mean $P \in \mathsf{Proc}\ \Gamma$. Similarly, actions are well-formed only in closing contexts; we write $a : \mathsf{Action}\ \Gamma$ to mean that Γ is closing for a, as defined below.

$$\boxed{\Gamma \vdash P}$$

$$\frac{}{\Gamma \vdash \mathbf{0}} \qquad \frac{\Gamma + 1 \vdash P}{\Gamma \vdash \underline{x}.P}\ x \in \Gamma \qquad \frac{\Gamma \vdash P}{\Gamma \vdash \overline{x}\langle y\rangle.P}\ x, y \in \Gamma \qquad \frac{\Gamma \vdash P \qquad \Gamma \vdash Q}{\Gamma \vdash P + Q}$$

$$\frac{\Gamma \vdash P \qquad \Gamma \vdash Q}{\Gamma \vdash P \mid Q} \qquad \frac{\Gamma + 1 \vdash P}{\Gamma \vdash \nu P} \qquad \frac{\Gamma \vdash P}{\Gamma \vdash !P}$$

$$\boxed{a : \mathsf{Action}\ \Gamma}$$

$$\frac{}{\underline{x} : \mathsf{Action}\ \Gamma}\ x \in \Gamma \qquad \frac{}{\overline{x} : \mathsf{Action}\ \Gamma}\ x \in \Gamma \qquad \frac{}{\overline{x}\langle y\rangle : \mathsf{Action}\ \Gamma}\ x, y \in \Gamma \qquad \frac{}{\tau : \mathsf{Action}\ \Gamma}$$

Fig. 1: Syntax of processes and actions

To specify the labelled transition semantics, it is convenient to distinguish *bound* actions b from non-bound actions c. A bound action $b : \mathsf{Action}\ \Gamma$ is of the form $\underline{x}$ or $\overline{x}$, and shifts a process from Γ to a target context $\Gamma + 1$, freeing the index 0. A non-bound action $c : \mathsf{Action}\ \Gamma$ is of the form $\overline{x}\langle y\rangle$ or τ, and has a target context which is also Γ. Meta-variable a ranges over all actions, bound and non-bound. $|a|$ denotes the amount by which the action increments the context; thus $|b| = 1$ and $|c| = 0$.

2.1. *Renamings*

A de Bruijn indices formulation of π-calculus makes extensive use of renamings. A *renaming* $\rho : \Gamma \longrightarrow \Delta$ is any function (injective or otherwise) from names in Γ to names in Δ. The labelled transition semantics makes use of the lifting of the successor function

$\cdot + 1$ on natural numbers to renamings, which we call push to avoid confusion with the $\cdot + 1$ operation on contexts; $\mathsf{pop}\ y$, which undoes the effect of push, replacing 0 by y; and swap, which transposes the roles of 0 and 1 but otherwise acts as the identity. This de Bruijn treatment of π-calculus is similar to that of Hirschkoff's asynchronous μs calculus [Hirschkoff 1999], except that we give a late rather than early semantics; other differences are discussed in § 5 below.

$$\boxed{\mathsf{push}_\Gamma : \Gamma \longrightarrow \Gamma + 1} \qquad \boxed{\mathsf{pop}_\Gamma\ y : \Gamma + 1 \longrightarrow \Gamma} \qquad \boxed{\mathsf{swap}_\Gamma : \Gamma + 2 \longrightarrow \Gamma + 2}$$

$$\mathsf{push}\ x = x + 1$$

$$\begin{aligned} \mathsf{pop}\ y\ 0 &= y \\ \mathsf{pop}\ y\ (x + 1) &= x \end{aligned}$$

$$\begin{aligned} \mathsf{swap}\ 0 &= 1 \\ \mathsf{swap}\ 1 &= 0 \\ \mathsf{swap}\ (x + 2) &= x + 2 \end{aligned}$$

Fig. 2: push, pop and swap renamings

The Γ subscripts that appear on push_Γ, $\mathsf{pop}_\Gamma\ y$ and swap_Γ are shown in grey to indicate that they may be omitted when their value is obvious or irrelevant; this is a convention we use throughout the paper.

2.1.1. *Lifting renamings to processes and actions* The functorial extension $\rho^* : \mathsf{Proc}\ \Gamma \longrightarrow \mathsf{Proc}\ \Delta$ of a renaming $\rho : \Gamma \longrightarrow \Delta$ to processes is defined in the usual way. Renaming under a binder utilises the action of $\cdot + 1$ on renamings, which is also functorial. Syntactically, ρ^* binds tighter than any process constructor, and $\cdot + 1$ has higher precedence than composition, so that (for example) $\mathsf{pop}\ 0 \circ \mathsf{push} + 1$ means $\mathsf{pop}\ 0 \circ (\mathsf{push} + 1)$, not $(\mathsf{pop}\ 0 \circ \mathsf{push}) + 1$.

$$\boxed{\cdot^* : (\Gamma \longrightarrow \Delta) \longrightarrow \mathsf{Proc}\ \Gamma \longrightarrow \mathsf{Proc}\ \Delta}$$

$$\begin{aligned} \rho^* \mathbf{0} &= \mathbf{0} \\ \rho^*(\underline{x}.P) &= \underline{\rho x}.(\rho + 1)^* P \\ \rho^*(\overline{x}\langle y \rangle.P) &= \overline{\rho x}\langle \rho y \rangle.\rho^* P \\ \rho^*(P + Q) &= \rho^* P + \rho^* Q \\ \rho^*(P \mid Q) &= \rho^* P \mid \rho^* Q \\ \rho^*(\nu P) &= \nu(\rho + 1)^* P \\ \rho^*(!P) &= !\rho^* P \end{aligned}$$

$$\boxed{\cdot^* : (\Gamma \longrightarrow \Delta) \longrightarrow \mathsf{Action}\ \Gamma \longrightarrow \mathsf{Action}\ \Delta}$$

$$\begin{aligned} \rho^*\ \underline{x} &= \underline{\rho x} \\ \rho^*\ \overline{x} &= \overline{\rho x} \\ \rho^*\ \tau &= \tau \\ \rho^*\ \overline{x}\langle y \rangle &= \overline{\rho x}\langle \rho y \rangle \end{aligned}$$

$$\boxed{\cdot + 1 : (\Gamma \longrightarrow \Delta) \longrightarrow \Gamma + 1 \longrightarrow \Delta + 1}$$

$$\begin{aligned} (\rho + 1)\ 0 &= 0 \\ (\rho + 1)\ (x + 1) &= \rho x + 1 \end{aligned}$$

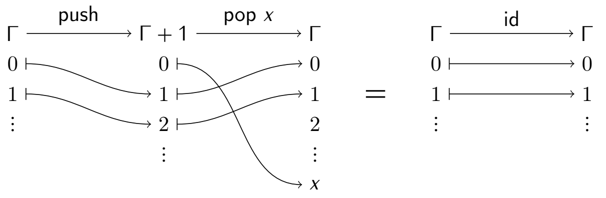
Fig. 3: Renaming for processes and actions

2.1.2. *Properties of renamings* Several equational properties of renamings are used throughout the development; here we present the ones mentioned elsewhere in the paper. For each lemma, we give the corresponding commutative diagram underneath on the left, along with a string diagram that offers a graphical intuition for why the lemma holds.

Lemma 1. $\mathsf{pop}\ x \circ \mathsf{push} = \mathsf{id}$

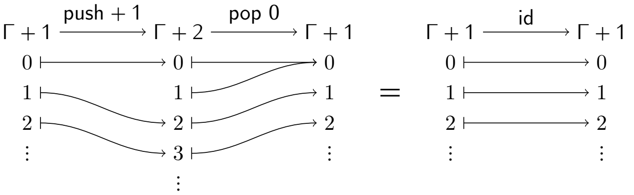
Freeing the index 0 and then immediately substituting x for it is a no-op.

$$\begin{tikzcd} \Gamma \arrow[r, "\mathsf{push}"] \arrow[dr, equal] & \Gamma+1 \arrow[d, "\mathsf{pop}\ x"] \\ & \Gamma \end{tikzcd}$$



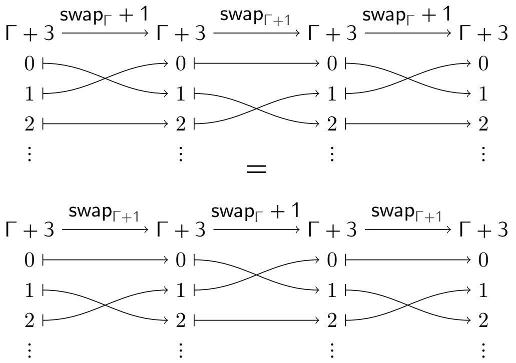
Lemma 2. $\mathsf{pop}\ 0 \circ \mathsf{push} + 1 = \mathsf{id}$

$$\begin{tikzcd} \Gamma+1 \arrow[r, "\mathsf{push}_\Gamma + 1"] \arrow[dr, equal] & \Gamma+2 \arrow[d, "\mathsf{pop}_{\Gamma+1}\ 0"] \\ & \Gamma+1 \end{tikzcd}$$



Lemma 3. $\mathsf{swap} + 1 \circ \mathsf{swap} \circ \mathsf{swap} + 1 = \mathsf{swap} \circ \mathsf{swap} + 1 \circ \mathsf{swap}$

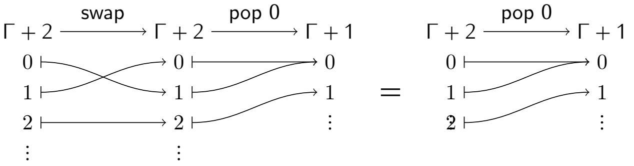
$$\begin{tikzcd} & \Gamma+3 \arrow[r, "\mathsf{swap}_\Gamma + 1"] & \Gamma+3 \arrow[dr, "\mathsf{swap}_{\Gamma+1}"] & \\ \Gamma+3 \arrow[ur, "\mathsf{swap}_{\Gamma+1}"] \arrow[dr, "\mathsf{swap}_\Gamma + 1"'] & & & \Gamma+3 \\ & \Gamma+3 \arrow[r, "\mathsf{swap}_{\Gamma+1}"'] & \Gamma+3 \arrow[ur, "\mathsf{swap}_\Gamma + 1"'] & \end{tikzcd}$$



The above are two ways to swap indices 0 and 2.

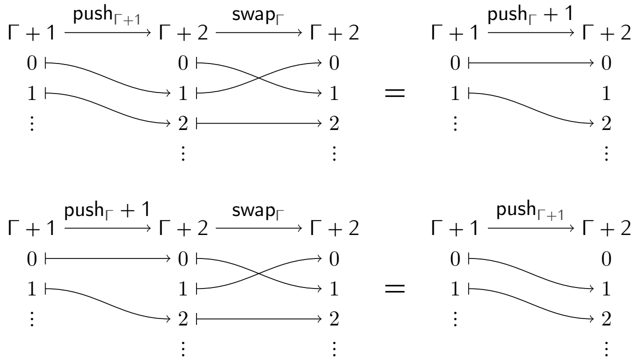
Lemma 4. $\mathsf{pop}\ 0 \circ \mathsf{swap} = \mathsf{pop}\ 0$

$$\begin{tikzcd} \Gamma+2 \arrow[r, shift left, "\mathsf{swap}"] \arrow[r, shift right, "\mathsf{id}"'] & \Gamma+2 \arrow[r, "\mathsf{pop}_{\Gamma+1}\ 0"] & \Gamma+1 \end{tikzcd}$$



Lemma 5. $\mathsf{swap} \circ \mathsf{push} + 1 = \mathsf{push}$, $\mathsf{swap} \circ \mathsf{push} = \mathsf{push} + 1$

$$\begin{tikzcd}
\Gamma + 1 \arrow[r, "\mathsf{push}_{\Gamma} + 1"] \arrow[dr, "\mathsf{push}_{\Gamma+1}"'] & \Gamma + 2 \arrow[d, shift left, "\mathsf{swap}"] \\
& \Gamma + 2 \arrow[u, shift left, "\mathsf{swap}"]
\end{tikzcd}$$


$\Gamma + 1 \xrightarrow{\mathsf{push}_{\Gamma+1}} \Gamma + 2 \xrightarrow{\mathsf{swap}_{\Gamma}} \Gamma + 2$
0
1
⋮
0
1
2
⋮
0
1
2
⋮
=
$\Gamma + 1 \xrightarrow{\mathsf{push}_{\Gamma} + 1} \Gamma + 2$
0
1
⋮
0
1
2
⋮
$\Gamma + 1 \xrightarrow{\mathsf{push}_{\Gamma} + 1} \Gamma + 2 \xrightarrow{\mathsf{swap}_{\Gamma}} \Gamma + 2$
0
1
⋮
0
1
2
⋮
0
1
2
⋮
=
$\Gamma + 1 \xrightarrow{\mathsf{push}_{\Gamma+1}} \Gamma + 2$
0
1
⋮
0
1
2
⋮

Lemma 6. $\mathsf{push} \circ \rho = \rho + 1 \circ \mathsf{push}$

Lemma 7. $\rho \circ \mathsf{pop}\ x = \mathsf{pop}\ \rho x \circ \rho + 1$

Lemma 8. $\mathsf{swap} \circ \rho + 2 = \rho + 2 \circ \mathsf{swap}$

These last three lemmas assert various naturality properties of push, $\mathsf{pop}\ x$ and swap.

$$\begin{tikzcd}
\Gamma \arrow[r, "\mathsf{push}_{\Gamma}"] \arrow[d, "\rho"'] & \Gamma + 1 \arrow[r, "\mathsf{pop}_{\Gamma}\ x"] \arrow[d, "\rho + 1"] & \Gamma \arrow[d, "\rho"] \\
\Delta \arrow[r, "\mathsf{push}_{\Delta}"'] & \Delta + 1 \arrow[r, "\mathsf{pop}_{\Delta}\ \rho x"'] & \Delta
\end{tikzcd}
\qquad
\begin{tikzcd}
\Gamma + 2 \arrow[r, "\mathsf{swap}_{\Gamma}"] \arrow[d, "\rho + 2"'] & \Gamma + 2 \arrow[d, "\rho + 2"] \\
\Delta + 2 \arrow[r, "\mathsf{swap}_{\Delta}"'] & \Delta + 2
\end{tikzcd}$$

2.2. *Labelled transition semantics*

An important feature of our presentation is that each transition rule has an explicit constructor name. This allow derivations to be written in a compact, expression-like form, similar to the *proven transitions* used by Boudol and Castellani [1989] to define notions of concurrency and residuation for CCS. However, rather than giving an additional inductive definition describing the structure of a "proof" that $P \xrightarrow{a} R$, we simply treat the inductive definition of $\xrightarrow{\quad}$ as a data type. This is a natural approach in a dependently-typed setting.

The rule names are summarised below, and have been chosen to reflect, where possible, the structure of the process triggering the rule. The corresponding relation $P \xrightarrow{a} R$ is defined in Figure 4, for any process $\Gamma \vdash P$, any $a : \mathsf{Action}\ \Gamma$ and any $\Gamma + |a| \vdash R$.

Transition	t, u ::=	$\underline{x}.P$	input on x
		$\overline{x}\langle y\rangle.P$	output y on x
		$t + Q \quad P + u$	choose left or right branch
		$t\ {}^{a}\!\mid Q \quad P \mid^{a} u$	propagate a through parallel composition on the left or right
		$t \mid_{y} u \quad t\ {}_{y}\!\mid u$	synchronise (receiving y on the left or right)
		$\overline{\nu} t$	initiate extrusion of ν
		$t \mid_{\nu} u \quad t\ {}_{\nu}\!\mid u$	ν-synchronise (receiving 0 on the left or right)
		$\nu^{a} t$	propagate a through binder
		$!t$	replicate

$$\boxed{P \xrightarrow{a} R}$$

$$\underline{x}.P \;\; \frac{}{\underline{x}.P \xrightarrow{\underline{x}} P} \qquad \overline{x}\langle y\rangle.P \;\; \frac{}{\overline{x}\langle y\rangle.P \xrightarrow{\overline{x}\langle y\rangle} P} \qquad \cdot + Q \;\; \frac{P \xrightarrow{a} R}{P + Q \xrightarrow{a} R}$$

$$\cdot\ {}^{c}|\ Q \;\; \frac{P \xrightarrow{c} R}{P \mid Q \xrightarrow{c} R \mid Q} \qquad \cdot\ {}^{b}|\ Q \;\; \frac{P \xrightarrow{b} R}{P \mid Q \xrightarrow{b} R \mid \mathsf{push}^{*}Q}$$

$$\cdot \mid_{y} \cdot \;\; \frac{P \xrightarrow{\underline{x}} R \qquad Q \xrightarrow{\overline{x}\langle y\rangle} S}{P \mid Q \xrightarrow{\tau} (\mathsf{pop}\ y)^{*}R \mid S} \qquad \overline{\nu}\cdot \;\; \frac{P \xrightarrow{\overline{(x+1)}\langle 0\rangle} R}{\nu P \xrightarrow{\overline{x}} R}$$

$$\cdot \mid_{\nu} \cdot \;\; \frac{P \xrightarrow{\underline{x}} R \qquad Q \xrightarrow{\overline{x}} S}{P \mid Q \xrightarrow{\tau} \nu(R \mid S)} \qquad \nu^{c}\cdot \;\; \frac{P \xrightarrow{\mathsf{push}^{*}c} R}{\nu P \xrightarrow{c} \nu R}$$

$$\nu^{b}\cdot \;\; \frac{P \xrightarrow{\mathsf{push}^{*}b} R}{\nu P \xrightarrow{b} \nu(\mathsf{swap}^{*}R)} \qquad !\cdot \;\; \frac{P \mid !P \xrightarrow{a} R}{!P \xrightarrow{a} R}$$

Fig. 4: Labelled transition rules ($P + \cdot$, $P\,|^{b}\,\cdot$, $P\,|^{c}\,\cdot$, $\cdot\,{}_{\nu}|\,\cdot$ and $\cdot\,{}_{y}|\,\cdot$ variants omitted)

The constructor name for each rule is shown to the left of the rule. There is an argument position, indicated by $\cdot$, for each premise of the rule. Note that there are two forms of the transition constructors $\cdot\,{}^{a}|\,\cdot$ and $\nu^{a}\cdot$ distinguished by whether they are indexed by a bound action b or by a non-bound action c. Omitted from Figure 4 are additional (but symmetric) rules of the form $P + \cdot$, $P\,|^{b}\,\cdot$ and $P\,|^{b}\,\cdot$ where the sub-transition occurs on the opposite side of the operator, and also $\cdot\,{}_{y}|\,\cdot$ (synchronise) and $\cdot\,{}_{\nu}|\,\cdot$ (ν-synchronise) rules in which the positions of sender and receiver are transposed. These are all straightforward variants of the rules shown, and are omitted from the figure to avoid clutter. Meta-variables t and u range over transition derivations; if $t : P \xrightarrow{a} R$ then $\mathsf{src}(t)$ denotes P and $\mathsf{tgt}(t)$ denotes R.

Although a de Bruijn formulation of pi calculus requires a certain amount of housekeeping, one pleasing consequence is that the usual side-conditions associated with the π-calculus transition rules are either subsumed by syntactic constraints on actions, or "operationalised" using the renamings above. In particular:

1 The use of push in the $\cdot\,{}^{b}|\ Q$ rule corresponds to the usual side-condition asserting that the binder being propagated by P is not free in Q. In the de Bruijn setting every binder "locally" has the name 0, and so this requirement can be operationalised by rewiring Q so that the name 0 is reserved. The push will be matched by a later pop which substitutes for 0, in the event that the action has a successful synchronisation.

2 The $\overline{\nu}\cdot$ rule requires an extrusion to be initiated by an output of the form $\overline{x+1}\langle 0\rangle$, capturing the usual side-condition that the name being extruded *on* is distinct from the name being extruded.

3 The rules of the form ν^{a} require that the action being propagated has the form $\mathsf{push}^* a$, ensuring that it contains no uses of index 0. This corresponds to the usual requirement that an action can only propagate through a binder that it does not mention.

The use of swap in the ν^{b} case follows Hirschkoff [1999] and has no counterpart outside of the de Bruijn setting. As a propagating binder passes through another binder, their local indices are 0 and 1. Propagation transposes the binders, and so to preserve naming we rewire R with a "braid" that swaps 0 and 1. Since binders are also reordered by *permutations* that relate causally equivalent executions, the swap renaming will also play an important role when we consider concurrent transitions (§ 3).

The following schematic derivation shows how the compact notation works. Suppose $t : P \xrightarrow{\overline{z+2}\langle 0\rangle} R$ takes place immediately under a ν-binder, causing the scope of the binder to be extruded. Then suppose the resulting bound output propagates through another binder, giving the partial derivation on the left:

$$\nu^{\overline{z}}\cdot\ \dfrac{\overline{\nu}\cdot\ \dfrac{t\ \dfrac{\vdots}{P \xrightarrow{\overline{z+2}\langle 0\rangle} R}}{\nu P \xrightarrow{\overline{z+1}} R}}{\nu\nu P \xrightarrow{\overline{z}} \nu R} \qquad \nu^{\overline{z}}\cdot\ \dfrac{\overline{\nu}t\ \dfrac{\vdots}{\nu P \xrightarrow{\overline{z+1}} R}}{\nu\nu P \xrightarrow{\overline{z}} \nu R} \qquad \nu^{\overline{z}}\overline{\nu}t\ \dfrac{\vdots}{\nu\nu P \xrightarrow{\overline{z}} \nu R}$$

with t standing in for the rest of the derivation. The constructors annotating the left-hand side of the derivation tree (shown in blue in the electronic version of this article) can be thought of as a partially unrolled "transition term" representing the proof. The $\cdot$ placeholders associated with each constructor are conceptually filled by the transition terms annotating the premises of that step. We can "roll up" the derivation by a single step, by moving the premises into their corresponding placeholders, as shown in the middle figure.

By repeating this process, we can write the whole derivation compactly as $\nu^{\overline{z}}\overline{\nu}t$, as shown on the right. Thus the compact form is simply a flattened transition derivation: similar to a simply-typed λ-calculus term written as a conventional expression, in a (Church-style) setting where a term is, strictly speaking, a typing derivation.

2.2.1. *Residuals of transitions and renamings* A transition t with action a survives any suitably-typed renaming ρ. Moreover ρ has an image in t, which is simply $\rho + |a|$.

Lemma 9. Suppose $t : P \xrightarrow{a} Q$ and $\rho : \Gamma \longrightarrow \Delta$, where $\Gamma \vdash P$. Then there exists a transition $\rho^* t : \rho^* P \xrightarrow{\rho^* a} (\rho + |a|)^* Q$.

$$\begin{tikzcd}
P \arrow[r, "t^{c}"] \arrow[d, "\rho^*"'] & Q \arrow[d, "\rho^*"] \\
\rho^* P \arrow[r, dotted, "(\rho^* t)^{\rho^* c}"'] & \rho^* Q
\end{tikzcd}
\qquad
\begin{tikzcd}
P \arrow[r, "t^{b}"] \arrow[d, "\rho^*"'] & Q \arrow[d, "(\rho + 1)^*"] \\
\rho^* P \arrow[r, dotted, "(\rho^* t)^{\rho^* b}"'] & (\rho + 1)^* Q
\end{tikzcd}$$

Proof. By the following defining equations. The various renaming lemmas needed to enable the induction hypothesis in each case are omitted.

$\boxed{\rho^* t^c}$

$$\begin{aligned}
\rho^*(\overline{x}\langle y\rangle.P) &= \overline{\rho x}\langle \rho y\rangle.\rho^* P\\
\rho^*(t + Q) &= \rho^* t + \rho^* Q\\
\rho^*(P + u) &= \rho^* P + \rho^* u\\
\rho^*(P \,|^{c}\, u) &= \rho^* P \,|^{\rho^* c}\, \rho^* u\\
\rho^*(t \,{}^{c}|\, Q) &= \rho^* t \,{}^{\rho^* c}|\, \rho^* Q\\
\rho^*(t \,|_{y}\, u) &= \rho^* t \,|_{\rho^* y}\, \rho^* u\\
\rho^*(t \,|_{\nu}\, u) &= \rho^* t \,|_{\nu}\, \rho^* u\\
\rho^*(\nu^{c} t) &= \nu^{\rho^* c}(\rho + 1)^* t\\
\rho^*(!t) &= !\rho^* t
\end{aligned}$$

$\boxed{\rho^* t^b}$

$$\begin{aligned}
\rho^*(\underline{x}.P) &= \underline{\rho x}.(\rho + 1)^* P\\
\rho^*(t + Q) &= \rho^* t + \rho^* Q\\
\rho^*(P + u) &= \rho^* P + \rho^* u\\
\rho^*(P \,|^{b}\, u) &= \rho^* P \,|^{\rho^* b}\, \rho^* u\\
\rho^*(t \,{}^{b}|\, Q) &= \rho^* t \,{}^{\rho^* b}|\, \rho^* Q\\
\rho^*(\overline{\nu} t) &= \overline{\nu}(\rho + 1)^* t\\
\rho^*(\nu^{b} t) &= \nu^{\rho^* b}(\rho + 1)^* t\\
\rho^*(!t) &= !\rho^* t
\end{aligned}$$

We would not expect $\rho^* t$ to be derivable for arbitrary ρ in all extensions of the π-calculus. In particular, the mismatch operator $[x \neq y]P$ that steps to P if x and y are distinct names is only stable under injective renamings.

2.2.2. *Structural congruences* Our LTS semantics is standard and therefore closed under the usual π-calculus congruences. Structural congruences can be formalised as a bisimulation, using an analogue of the notion of residuation with respect to a transition used elsewhere in this paper. This remains out of scope of the present development.

3. Concurrency, residuals and cofinality

Transitions $P \xrightarrow{a} R$ and $Q \xrightarrow{a'} S$ are *coinitial* when $P = Q$. In this section we formalise a symmetric, irreflexive *concurrency* relation $\smile$ over coinitial transitions. Concurrent transitions $t \smile t'$ are independent, or causally unordered. In an interleaving semantics, t and t' can execute in either order without significant interference; in a true concurrency setting, t and t' form a single, two-dimensional "parallel move" [Curry and Feys 1958]. Concurrency was explored notably by Lévy [1980] for the λ-calculus, and later by Stark [1989] for arbitrary transition systems. The inspiration for the treatment presented here is Boudol and Castellani's concurrency relation for CCS [1989].

$$\begin{tikzcd}
 & R \arrow[dr, dotted, "t'/t"] & \\
P \arrow[ur, "t"] \arrow[dr, "t'"'] & & Q \\
 & R' \arrow[ur, dotted, "t/t'"'] &
\end{tikzcd}$$

Fig. 5: Conventional diamond property for $t \smile t'$

The essence of $t \smile t'$ is illustrated in Figure 5. If either execution step is taken, the other remains valid, and moreover once both are taken, one ends up in (essentially) the same state, regardless of which step is taken first. However, concurrent transitions are not completely independent: the location and indeed the nature of the redex acted on by one transition may change as a consequence of the earlier transition being taken. This idea is captured by the *residual* t'/t ("t' after t"), which specifies how t' must be transformed to operate on $\mathtt{tgt}(t)$ (sometimes called *pseudocommutation* [Angiuli et al. 2014]).

The requirement that t'/t and t/t' are *cofinal* – have the same target state – is straightforward when the transitions preserve the free variables of a term. This is trivially the case in CCS since there are no binders, and is also true of the λ-calculus, where reductions are usually defined on closed terms. In the late-style π-calculus that we consider here, there are transition rules that "open" a process with respect to a name, with the action on the transition representing the upwards propagation of the binder through the process term. In this setting the notion of cofinality is non-trivial; de Bruijn indices make this subtlety more explicit. We discuss this, with examples, in this section. Permutation of concurrent transitions induces a congruence on traces called *causal equivalence*, which we turn to in § 4.

3.1. *Concurrent transitions*

In our setting, a transition $t : P \xrightarrow{a} R$ is a *proof* that locates a redex in P, witnessing the fact that $(P, a, R) \in \xrightarrow{\quad}$. The concurrency relation $\smile$ relates two such proofs; it is defined as the symmetric closure of the relation defined inductively by the rules in Figure 6. The figure makes use of the compact notation for transitions introduced in § 2.2. As before, trivial variants of the rules are omitted for clarity. For the transition constructors of the form $\cdot\ {}^{a}|\ Q$ and $\nu^{a}\cdot$ that come in bound and non-bound variants, we abuse notation a little and write a single $\smile$ rule quantified over a to mean that there are two separate (but otherwise identical) cases.

Intuitively, transitions are concurrent when they pick out non-overlapping redexes. The only axiom, $P\,|^{a}\ u \smile t\ {}^{a'}|\ Q$, says that two transitions t and u are concurrent if they take place on opposite sides of a parallel composition. The remaining rules propagate concurrent sub-transitions up through restriction, choice, parallel composition, and replication. There are no rules allowing us to conclude that a transition which takes the left branch of a choice is concurrent with transition which takes the right branch of the same choice; choices are mutually exclusive. Likewise, there are no rules allowing us to conclude that an input or output transition is concurrent with any other transition. Since t and t' are coinitial, if one of them picks out a prefix then the other necessarily picks out the same prefix, and so they are equal and thus not concurrent.

The $t\,|_{y}\ u \smile t'\,|_{z}\ u'$ rule says that a synchronisation is concurrent with another, as long as the two input transitions t and t' are concurrent on the left branch of the parallel composition, and the two output transitions u and u' are concurrent on the right. The $t\,|_{y}\ u \smile t'\ {}_{z}|\ u'$ variant is similar, but permits concurrent input and output transitions on the left, with their respective synchronisation partners concurrent on the right. The $t\,|_{y}\ u \smile t'\ {}_{\nu}|\ u'$ rule and variants are analogous, but permit a plain synchronisation to be concurrent with a ν-synchronisation. The main result of this section is that the concurrency relation captured by $\smile$ is sound up to a suitable notion of cofinality.

Example 1 (Concurrent transitions). Consider the π-calculus process (using conventional

$\boxed{t \smile t'}$

$$\dfrac{}{P \mid^{a} u \smile t\,{}^{a'}\!\mid Q} \qquad \dfrac{t \smile t'}{t\,{}^{a}\!\mid Q \smile t' \mid_{y} u} \qquad \dfrac{t \smile t'}{t\,{}^{a}\!\mid Q \smile t'\,{}_{y}\!\mid u} \qquad \dfrac{u \smile u'}{P \mid^{a} u \smile t \mid_{y} u'}$$

$$\dfrac{u \smile u'}{P \mid^{a} u \smile t\,{}_{y}\!\mid u'} \qquad \dfrac{t \smile t'}{t\,{}^{a}\!\mid Q \smile t' \mid_{\nu} u} \qquad \dfrac{t \smile t'}{t\,{}^{a}\!\mid Q \smile t'\,{}_{\nu}\!\mid u} \qquad \dfrac{u \smile u'}{P \mid^{a} u \smile t \mid_{\nu} u'}$$

$$\dfrac{u \smile u'}{P \mid^{a} u \smile t\,{}_{\nu}\!\mid u'} \qquad \dfrac{t \smile t'}{t + Q \smile t' + Q} \qquad \dfrac{u \smile u'}{P + u \smile P + u'} \qquad \dfrac{t \smile t'}{P \mid^{a} t \smile P \mid^{a'} t'}$$

$$\dfrac{t \smile t'}{t\,{}^{a}\!\mid Q \smile t'\,{}^{a'}\!\mid Q} \qquad \dfrac{t \smile t' \qquad u \smile u'}{t \mid_{y} u \smile t' \mid_{z} u'} \qquad \dfrac{t \smile t' \qquad u \smile u'}{t\,{}_{y}\!\mid u \smile t'\,{}_{z}\!\mid u'} \qquad \dfrac{t \smile t' \qquad u \smile u'}{t \mid_{y} u \smile t'\,{}_{z}\!\mid u'}$$

$$\dfrac{t \smile t' \qquad u \smile u'}{t \mid_{y} u \smile t' \mid_{\nu} u'} \qquad \dfrac{t \smile t' \qquad u \smile u'}{t\,{}_{y}\!\mid u \smile t' \mid_{\nu} u'} \qquad \dfrac{t \smile t' \qquad u \smile u'}{t \mid_{y} u \smile t'\,{}_{\nu}\!\mid u'} \qquad \dfrac{t \smile t' \qquad u \smile u'}{t\,{}_{y}\!\mid u \smile t'\,{}_{\nu}\!\mid u'}$$

$$\dfrac{t \smile t' \qquad u \smile u'}{t \mid_{\nu} u \smile t' \mid_{\nu} u'} \qquad \dfrac{t \smile t' \qquad u \smile u'}{t\,{}_{\nu}\!\mid u \smile t'\,{}_{\nu}\!\mid u'} \qquad \dfrac{t \smile t' \qquad u \smile u'}{t \mid_{\nu} u \smile t'\,{}_{\nu}\!\mid u'} \qquad \dfrac{t \smile t'}{\overline{\nu} t \smile \overline{\nu} t'}$$

$$\dfrac{t \smile t'}{\overline{\nu} t \smile \nu^{a} t'} \qquad \dfrac{t \smile t'}{\nu^{a} t \smile \nu^{a'} t'} \qquad \dfrac{t \smile t'}{!t \smile !t'}$$

Fig. 6: Concurrent transitions

named syntax) $(\nu y)\, \overline{x}\langle y\rangle.P \mid \overline{z}\langle y\rangle.Q$. This can take two transitions, the first one sending y on x, resulting in $P \mid \overline{z}\langle y\rangle.Q$, and the second one sending y on channel z, resulting in $\overline{x}\langle y\rangle.P \mid Q$. Notice that y becomes free in both processes.

In de Bruijn notation, this process is written $\nu(\overline{x+1}\langle 0\rangle.P \mid \overline{z+1}\langle 0\rangle.Q)$. It can take two transitions, each resulting in an extrusion of the ν-binder; call these t and t'. The transition t initiates the extrusion $\overline{x}$ on the left branch of the parallel composition:

$$\overline{\nu}\cdot\ \dfrac{\cdot\ {}^{\overline{x+1}\langle 0\rangle}\!\mid \overline{z+1}\langle 0\rangle.Q\ \ \dfrac{\Gamma + 1 \vdash \overline{x+1}\langle 0\rangle.P \xrightarrow{\overline{x+1}\langle 0\rangle} \Gamma + 1 \vdash P}{\Gamma + 1 \vdash \overline{x+1}\langle 0\rangle.P \mid \overline{z+1}\langle 0\rangle.Q \xrightarrow{\overline{x+1}\langle 0\rangle} \Gamma + 1 \vdash P \mid \overline{z+1}\langle 0\rangle.Q}}{\Gamma \vdash \nu(\overline{x+1}\langle 0\rangle.P \mid \overline{z+1}\langle 0\rangle.Q) \xrightarrow{\overline{x}} \Gamma + 1 \vdash P \mid \overline{z+1}\langle 0\rangle.Q}$$

The transition t' initiates an extrusion $\overline{z}$ of the same binder on the right branch of the parallel composition:

$$\overline{\nu}\cdot\ \dfrac{\overline{x+1}\langle 0\rangle.Q \mid^{\overline{z+1}\langle 0\rangle} \cdot\ \ \dfrac{\Gamma + 1 \vdash \overline{z+1}\langle 0\rangle.Q \xrightarrow{\overline{z+1}\langle 0\rangle} \Gamma + 1 \vdash Q}{\Gamma + 1 \vdash \overline{x+1}\langle 0\rangle.P \mid \overline{z+1}\langle 0\rangle.Q \xrightarrow{\overline{z+1}\langle 0\rangle} \Gamma + 1 \vdash \overline{x+1}\langle 0\rangle.P \mid Q}}{\Gamma \vdash \nu(\overline{x+1}\langle 0\rangle.P \mid \overline{z+1}\langle 0\rangle.Q) \xrightarrow{\overline{z}} \Gamma + 1 \vdash \overline{x+1}\langle 0\rangle.P \mid Q}$$

Since the two transitions are coinitial and arise on opposite sides of a parallel composition, we can conclude $t \smile t'$ using the rules in Figure 6. Here is the proof, writing the derivations t and t' above using the compact notation for transitions:

$$\frac{\overline{x+1}\langle 0\rangle.P\ {}^{\overline{x+1}\langle 0\rangle}|\ \overline{z+1}\langle 0\rangle.Q \ \smile\ \overline{x+1}\langle 0\rangle.P\ |^{\overline{z+1}\langle 0\rangle}\ \overline{z+1}\langle 0\rangle.Q}{\overline{\nu}(\overline{x+1}\langle 0\rangle.P\ {}^{\overline{x+1}\langle 0\rangle}|\ \overline{z+1}\langle 0\rangle.Q) \ \smile\ \overline{\nu}(\overline{x+1}\langle 0\rangle.P\ |^{\overline{z+1}\langle 0\rangle}\ \overline{z+1}\langle 0\rangle.Q)}$$

■

3.2. *Residuals of concurrent transitions*

If two transitions are concurrent then their respective *residuals* provide a canonical way of merging or reconciling them.

Definition 1 (Residual t/t'). For any $t \smile t'$, the *residual* of t after t', written t/t', is defined by the equations in Figure 7.

$\boxed{t/t'}$

$$\begin{aligned}
(P\ |^{a}\ u)/(t\ {}^{c}|\ Q) &= \mathsf{tgt}(t)\ |^{a}\ u\\
(P\ |^{a}\ u)/(t\ {}^{b}|\ Q) &= \mathsf{tgt}(t)\ |^{a}\ \mathsf{push}^{*}u\\
(t\ {}^{a}|\ Q)/(P\ |^{c}\ u) &= t\ {}^{a}|\ \mathsf{tgt}(u)\\
(t\ {}^{a}|\ Q)/(P\ |^{b}\ u) &= \mathsf{push}^{*}t\ {}^{a}|\ \mathsf{tgt}(u)\\
(t\ {}^{a}|\ Q)/(t'\ |_{y}\ u) &= (\mathsf{pop}\ y)^{*}(t/t')\ {}^{a}|\ \mathsf{tgt}(u)\\
(P\ |^{a}\ u)/(t\ |_{y}\ u') &= (\mathsf{pop}\ y)^{*}\mathsf{tgt}(t)\ |^{a}\ u/u'\\
(t\ |_{y}\ u)/(t'\ {}^{b}|\ Q) &= t/t'\ |_{y}\ \mathsf{push}^{*}u\\
(t\ |_{y}\ u)/(t'\ {}^{c}|\ Q) &= t/t'\ |_{y}\ u\\
(t\ |_{y}\ u)/(P\ |^{b}\ u') &= \mathsf{push}^{*}t\ |_{y}\ u/u'\\
(t\ |_{y}\ u)/(P\ |^{c}\ u') &= t\ |_{y}\ u/u'\\
(t\ {}^{\underline{x}}|\ Q)/(t'\ |_{\nu}\ u) &= \nu^{\underline{x}}(t/t'\ {}^{\underline{x+1}}|\ \mathsf{tgt}(u))\\
(t\ {}^{\overline{x}}|\ Q)/(t'\ |_{\nu}\ u) &= \overline{\nu}(t/t'\ {}^{\overline{x+1}\langle 0\rangle}|\ \mathsf{tgt}(u))\\
(t\ {}^{c}|\ Q)/(t'\ |_{\nu}\ u) &= \nu^{c}(t/t'\ {}^{\mathsf{push}^{*}c}|\ \mathsf{tgt}(u))\\
(P\ |^{\underline{x}}\ u)/(t\ |_{\nu}\ u') &= \nu^{\underline{x}}(\mathsf{tgt}(t)\ |^{\underline{x+1}}\ u/u')\\
(P\ |^{\overline{x}}\ u)/(t\ |_{\nu}\ u') &= \overline{\nu}(\mathsf{tgt}(t)\ |^{\overline{x+1}\langle 0\rangle}\ u/u')\\
(P\ |^{c}\ u)/(t\ |_{\nu}\ u') &= \nu^{c}(\mathsf{tgt}(t)\ |^{\mathsf{push}^{*}c}\ u/u')\\
(t\ |_{\nu}\ u)/(t'\ {}^{b}|\ Q) &= t/t'\ |_{\nu}\ \mathsf{push}^{*}u\\
(t\ |_{\nu}\ u)/(t'\ {}^{c}|\ Q) &= t/t'\ |_{\nu}\ u\\
(t\ |_{\nu}\ u)/(P\ |^{\underline{x}}\ u') &= \mathsf{push}^{*}t\ |_{\nu}\ u/u'\\
(t\ |_{\nu}\ u)/(P\ |^{\overline{x}}\ u') &= \mathsf{push}^{*}t\ |_{0}\ u/u'\\
(t\ |_{\nu}\ u)/(P\ |^{c}\ u') &= t\ |_{\nu}\ u/u'\\
(t+Q)/(t'+Q) &= t/t'\\
(P\ |^{\underline{x}}\ u)/(P\ |^{b}\ u') &= \mathsf{push}^{*}P\ |^{\underline{x}}\ u/u'
\end{aligned}$$

$$\begin{aligned}
(P\ |^{b}\ u)/(P\ |^{\underline{x}}\ u') &= \mathsf{push}^{*}P\ |^{b}\ u/u'\\
(P\ |^{\overline{x}}\ u)/(P\ |^{\overline{u}}\ u') &= \mathsf{push}^{*}P\ |^{\overline{x+1}\langle 0\rangle}\ u/u'\\
(P\ |^{c}\ u)/(P\ |^{b}\ u') &= \mathsf{push}^{*}P\ |^{c}\ u/u'\\
(P\ |^{a}\ u)/(P\ |^{c}\ u') &= P\ |^{a}\ u/u'\\
(t\ {}^{\underline{x}}|\ Q)/(t'\ {}^{b}|\ Q) &= t/t'\ {}^{\underline{x}}|\ \mathsf{push}^{*}Q\\
(t\ {}^{b}|\ Q)/(t'\ {}^{\underline{x}}|\ Q) &= t/t'\ {}^{b}|\ \mathsf{push}^{*}Q\\
(t\ {}^{\overline{x}}|\ Q)/(t'\ {}^{\overline{u}}|\ Q) &= t/t'\ {}^{\overline{x+1}\langle 0\rangle}|\ \mathsf{push}^{*}Q\\
(t\ {}^{c}|\ Q)/(t'\ {}^{b}|\ Q) &= t/t'\ {}^{c}|\ \mathsf{push}^{*}Q\\
(t\ {}^{a}|\ Q)/(t'\ {}^{c}|\ Q) &= t/t'\ {}^{a}|\ Q\\
(t\ |_{y}\ u)/(t'\ |_{z}\ u') &= (\mathsf{pop}\ z)^{*}(t/t')\ |_{y}\ u/u'\\
(t\ |_{y}\ u)/(t'\ |_{\nu}\ u') &= \nu^{\tau}(t/t'\ |_{y}\ u/u')\\
(t\ |_{\nu}\ u)/(t'\ |_{z}\ u') &= (\mathsf{pop}\ z)^{*}(t/t')\ |_{\nu}\ u/u'\\
(t\ |_{\nu}\ u)/(t'\ |_{\nu}\ u') &= \nu^{\tau}(t/t'\ |_{\nu}\ u/u')\\
(\overline{\nu}t)/(\overline{\nu}t') &= t/t'\\
(\overline{\nu}t)/(\nu^{b}t') &= \overline{\nu}\ \mathsf{swap}^{*}(t/t')\\
(\overline{\nu}t)/(\nu^{c}t') &= \overline{\nu}\ t/t'\\
(\nu^{b}t)/(\overline{\nu}t') &= t/t'\\
(\nu^{c}t)/(\overline{\nu}t') &= t/t'\\
(\nu^{b}t)/(\nu^{b}t') &= \nu\ t/t'\\
(\nu^{c}t)/(\nu^{b}t') &= \nu^{c}\ \mathsf{swap}^{*}(t/t')\\
(\nu^{b}t)/(\nu^{c}t') &= \nu^{b}\ t/t'\\
(\nu^{c}t)/(\nu^{c}t') &= \nu^{c}\ t/t'\\
(!t)/(!t') &= t/t'
\end{aligned}$$

Fig. 7: Residual of t after t', omitting $\cdot\ {}_{y}|\ \cdot$ and $\cdot\ {}_{\nu}|\ \cdot$ cases

The above definition is a total and terminating function on concurrent transitions (in Agda, this is verified by the typechecker). Syntactically, the operator $\cdot/\cdot$ has higher precedence than any transition constructor. The definition makes use of the renaming lemmas in § 2.1.2 and the fact that the transition system is closed under renamings (Lemma 9).

Example 2 (Residuals of concurrent transitions). First recall the named process in Example 1 above, that is, $(\nu y)\ \overline{x}\langle y\rangle.P \mid \overline{z}\langle y\rangle.Q$. Both of the transitions it can perform are bound transitions, extruding y, which is no longer bound in the resulting process. After the first transition, the second can be performed as a free send of y along z and vice versa, and in both cases we obtain the process $P \mid Q$, again containing y free.

These observations are reflected in the de Bruijn representation. Since t and t' are concurrent there should exist residual transitions denoted t'/t and t/t', which are cofinal, allowing us to complete the square

$$\begin{tikzcd}
 & \Gamma+1 \vdash P \mid \overline{z+1}\langle 0\rangle.Q \arrow[dr, dotted, "t'/t"] & \\
\Gamma \vdash \nu(\overline{x+1}\langle 0\rangle.P \mid \overline{z+1}\langle 0\rangle.Q) \arrow[ur, "t"] \arrow[dr, "t'"'] & & \Gamma+\Delta \vdash R \\
 & \Gamma+1 \vdash \overline{x+1}\langle 0\rangle.P \mid Q \arrow[ur, dotted, "t/t'"'] &
\end{tikzcd}$$

for some $\Delta \in \{1,2\}$ and some process R. In the upper state $\mathsf{tgt}(t)$, the only candidate for t'/t is the output prefix $\overline{z+1}\langle 0\rangle$. However, the ν-binder to which index 0 refers no longer appears in $\mathsf{tgt}(t)$. Rather, that binder is propagating and index 0 is free, reflected by $\mathsf{tgt}(t)$ being in context $\Gamma+1$. When the output transition is taken, $\overline{z+1}\langle 0\rangle$ therefore simply propagates as a non-bound action, rather than causing a further extrusion:

$$P \mid^{\overline{z+1}\langle 0\rangle} \cdot \quad \frac{\Gamma+1 \vdash \overline{z+1}\langle 0\rangle.Q \xrightarrow{\overline{z+1}\langle 0\rangle} \Gamma+1 \vdash Q}{\Gamma+1 \vdash P \mid \overline{z+1}\langle 0\rangle.Q \xrightarrow{\overline{z+1}\langle 0\rangle} \Gamma+1 \vdash P \mid Q}$$

From the lower state $\mathsf{tgt}(t')$ the only candidate for t/t' is the output prefix $\overline{x+1}\langle 0\rangle$, and similar reasoning applies. Thus for t/t' we have

$$\cdot\ ^{\overline{x+1}\langle 0\rangle}\!\mid Q \quad \frac{\Gamma+1 \vdash \overline{x+1}\langle 0\rangle.P \xrightarrow{\overline{x+1}\langle 0\rangle} \Gamma+1 \vdash P}{\Gamma+1 \vdash \overline{x+1}\langle 0\rangle.P \mid Q \xrightarrow{\overline{x+1}\langle 0\rangle} \Gamma+1 \vdash P \mid Q}$$

and therefore $\Delta = 1$ and $R = P \mid Q$. In summary when concurrent t and t' extrude the same binder, their respective residuals are plain outputs, not bound outputs, because a given binder can only be extruded once.

To relate this example to the defining equations of $\cdot/\cdot$ we use the compact presentations of t and t' from the end of Example 1. It is then easy to see that the rules in Figure 7 indeed compute (in compact form) the derivation above for t/t':

$$\begin{array}{rl}
 & \overline{\nu}(\overline{x+1}\langle 0\rangle.P\ ^{\overline{x+1}\langle 0\rangle}\!\mid \overline{z+1}\langle 0\rangle.Q)/\overline{\nu}(\overline{x+1}\langle 0\rangle.P \mid^{\overline{z+1}\langle 0\rangle} \overline{z+1}\langle 0\rangle.Q) \\
= & (\overline{x+1}\langle 0\rangle.P\ ^{\overline{x+1}\langle 0\rangle}\!\mid \overline{z+1}\langle 0\rangle.Q)/(\overline{x+1}\langle 0\rangle.P \mid^{\overline{z+1}\langle 0\rangle} \overline{z+1}\langle 0\rangle.Q) \\
= & \overline{x+1}\langle 0\rangle.P\ ^{\overline{x+1}\langle 0\rangle}\!\mid Q
\end{array}$$

and similarly for t'/t. ■

Example 1 illustrated the basic idea of residuation, focusing on the specific case where the residuals of transitions with bound actions have actions that are not bound, a subtlety of residuation particular to π-calculus first noted by Cristescu et al. [2013]. To capture this and other aspects of residuation, it is useful to define a datatype of *concurrent actions* $a \smile a'$ and an associated notion of residual action a/a' and use these to index concurrent transitions and their residuals.

We define both of these using the diagrams in Figure 8 below. The datatype of concurrent actions, ranged over by $\mathring{a}$, has five constructors, one for each diagram; the arrows diverging on the left represent the concurrent actions a and a', and the arrows converging on the right define the corresponding residuals a'/a and a/a'. Beneath each diagram is the *braiding* relation $\mathrel{\overline{\underline{\times}}}_{\mathring{a}}$ which constitutes the notion of cofinality induced by that form of concurrent action.

$$\begin{tikzcd}
 & \text{(i)} & \\
 & \Gamma \arrow[dr, dotted, "c'"] & \\
\Gamma \arrow[ur, "c"] \arrow[dr, "c'"'] & & \Gamma \\
 & \Gamma \arrow[ur, dotted, "c"'] & \\
 & = &
\end{tikzcd}$$

$$\begin{tikzcd}
 & \text{(ii)} & \\
 & \Gamma + 1 \arrow[dr, dotted, "\mathsf{push}^{*} c"] & \\
\Gamma \arrow[ur, "b"] \arrow[dr, "c"'] & & \Gamma + 1 \\
 & \Gamma \arrow[ur, dotted, "b"'] & \\
 & = &
\end{tikzcd}$$

$$\begin{tikzcd}
 & \text{(iii)} & \\
 & \Gamma + 1 \arrow[dr, dotted, "\mathsf{push}^{*} b'"] & \\
\Gamma \arrow[ur, "b"] \arrow[dr, "b'"'] & & \Gamma + 2 \\
 & \Gamma + 1 \arrow[ur, dotted, "\mathsf{push}^{*} b"'] & \\
 & \mathrel{\underline{\times}} &
\end{tikzcd}$$

$$\begin{tikzcd}
 & \text{(iv)} & \\
 & \Gamma + 1 \arrow[dr, dotted, "\overline{y+1}\langle 0\rangle"] & \\
\Gamma \arrow[ur, "\overline{x}"] \arrow[dr, "\overline{y}"'] & & \Gamma + 1 \\
 & \Gamma + 1 \arrow[ur, dotted, "\overline{x+1}\langle 0\rangle"'] & \\
 & = &
\end{tikzcd}$$

$$\begin{tikzcd}
 & \text{(v)} & \\
 & \Gamma \arrow[dr, dotted, "\tau_{\nu}"] & \\
\Gamma \arrow[ur, "\tau_{\nu}"] \arrow[dr, "\tau_{\nu}"'] & & \Gamma \\
 & \Gamma \arrow[ur, dotted, "\tau_{\nu}"'] & \\
 & \mathrel{\overline{\times}} &
\end{tikzcd}$$

Fig. 8: Concurrent actions $\mathring{a} : a \smile a'$, residuals a'/a and a/a', and braiding relation $\mathrel{\overline{\underline{\times}}}_{\mathring{a}}$

Diagrams (i) and (ii) capture the general pattern when at most one of the actions is bound. In (ii), image of an action in a bound action is the original action shifted under a binder; in both cases cofinality is simply equality. Diagram (iii) is the general pattern when both actions are bound: in this case the target states P and P' are related by a "free braid" $P \mathrel{\underline{\times}} P'$ in the form of the permutation swap which renames 0 to 1 and 1 to 0, reflecting the transposition of the two binders.

Diagram (iv) and (v) are specific to name extrusion. Diagram (iv) is an exception to (iii) where the two bound actions happen to be extrusions of the same binder, as in Example 1; the $\overline{\nu}t \smile \overline{\nu}t'$ rule is the only concurrency rule that generates concurrent actions of this form. Diagram (v) is an exception to (i) where the two non-bound actions happen to be ν-synchronisations of distinct binders. In this case the residual actions will also be ν-synchronisations (as suggested by the informal τ_{ν} notation) and the target states are related by a "bound braid" $P \mathrel{\overline{\times}} P'$, essentially a free braid which has been "closed" by a pair of ν-binders, representing the transposition of those binders. The four variants of the $t \mid_{\nu} u \smile t' \mid_{\nu} u'$ rule generate concurrent actions of this form whenever the extruding binders are distinct.

Free and bound braids are now defined more formally.

Definition 2 (Free braid). For any processes $\Gamma + 2 \vdash P, R$ define the symmetric relation $P \underline{\times} R$ as follows. The context Γ is left implicit.

$$P \underline{\times} R \quad \Leftrightarrow \quad P = \mathsf{swap}^* R$$

Symmetry of $\underline{\times}$ is immediate from the involutivity of swap. Note that $\underline{\times}$ is not irreflexive, since $\mathsf{swap}^* P = P$ iff indices 0 and 1 are both unused in P.

Definition 3 (Bound braid). For any processes $\Gamma \vdash P, R$ inductively define the symmetric relation $P \overline{\times} R$ using the rules in Figure 9. Again the context Γ is left implicit.

$$\boxed{P \overline{\times} R}$$

$$\nu\nu\text{-}\mathsf{swap}_P \ \frac{P \underline{\times} P'}{\nu\nu P \overline{\times} \nu\nu P'} \qquad \cdot + Q \ \frac{P \overline{\times} R}{P + Q \overline{\times} R + Q} \qquad P + \cdot \ \frac{Q \overline{\times} S}{P + Q \overline{\times} P + S}$$

$$\cdot \mid Q \ \frac{P \overline{\times} R}{P \mid Q \overline{\times} R \mid Q} \qquad P \mid \cdot \ \frac{Q \overline{\times} S}{P \mid Q \overline{\times} P \mid S} \qquad \nu\cdot \ \frac{P \overline{\times} R}{\nu P \overline{\times} \nu R} \qquad !\cdot \ \frac{P \overline{\times} R}{!P \overline{\times} !R}$$

Fig. 9: Bound braid $P \overline{\times} R$

We adopt a compact term-like notation for $\overline{\times}$ proofs similar to the convention introduced earlier for transitions. As before, rule names are shown to the left of each rule, in blue. The symmetry of $\underline{\times}$ follows easily from the symmetry of $\overline{\times}$; moreover $\overline{\times}$ is also not irreflexive, because $\underline{\times}$ is not irreflexive. Meta-variables ϕ and ψ range over bound braids; $\mathsf{src}(\phi)$ and $\mathsf{tgt}(\phi)$ denote P and R for any $\phi : P \overline{\times} R$. Bound braids are "unobservable" in the sense that two processes related by a bound braid are strongly bisimilar. Indeed $\nu\nu(\mathsf{swap}^* P) \cong \nu\nu P$ is simply the de Bruijn counterpart of the familiar congruence $(\nu x y)\ P \cong (\nu y x)\ P$.

Concurrent actions and action residuals give transition residuals a more precise type (omitted for simplicity from the definitions of $t \smile t'$ and t/t'), making them somewhat easier to formally define. But more important here is how they determine the appropriate notion of cofinality relating $\mathsf{tgt}(t'/t)$ and $\mathsf{tgt}(t/t')$, namely the braiding relation $\overline{\underline{\times}}_{a,a'}$ specified beneath each diagram in Figure 8. A braiding relation is a singleton type, whose unique inhabitant precisely captures precisely the "rewiring" effect of reordering transitions that involve binders.

Definition 4 (Braiding). For any context Γ, any $a, a' \in \mathsf{Action}\ \Gamma$ and any $\mathring{a} : a \smile a'$, define the following symmetric relation $\overline{\underline{\times}}_{\mathring{a}}$ over processes in Γ', where Γ' is the target context of $\mathring{a}$.

$$\overline{\underline{\times}}_{\mathring{a}} \overset{\text{def}}{=} \text{one of } \underline{\times},\ \overline{\times} \text{ or } = \text{ as defined in Figure 8}$$

Our key soundness result is that the targets of the residuals of concurrent transitions $t \smile t'$ with actions $\mathring{a} : a \smile a'$ are cofinal in the sense of being related by $\overline{\underline{\times}}_{\mathring{a}}$. We need first that bound braids are closed under renamings, which we capture as a notion of residuation ϕ/ρ. The other residual ρ/ϕ is always ρ.

Lemma 10. For any $\Gamma \vdash P$, suppose $\phi : P \mathbin{\bar{\times}} Q$ and $\rho : \Gamma \longrightarrow \Delta$. Then there exists a bound braid $\phi/\rho : \rho^* P \mathbin{\bar{\times}} \rho^* Q$.

$$\begin{tikzcd}
P \arrow[r, "\phi"] \arrow[d, "\rho^*"'] & Q \arrow[d, "\rho^*"] \\
\rho^* P \arrow[r, dotted, "\phi/\rho"'] & \rho^* Q
\end{tikzcd}$$

Theorem 1 (Cofinality of residuals).

$$\begin{tikzcd}
 & \Gamma' \vdash R \arrow[r, dotted, "t'/t"] & \Gamma + \Delta \vdash Q \arrow[dd, dotted, no head, "\gamma_{t,t'}"] \\
\Gamma \vdash P \arrow[ur, "t"] \arrow[dr, "t'"'] & & \\
 & \Gamma'' \vdash R' \arrow[r, dotted, "t/t'"'] & \Gamma + \Delta \vdash Q'
\end{tikzcd}$$

Suppose $t \smile t'$ with actions $\mathring{a} : a \smile a'$. Then there exists a unique $\gamma_{t,t'} : \mathsf{tgt}(t'/t) \mathbin{\underline{\bar{\times}}}_{\mathring{a}} \mathsf{tgt}(t/t')$.

We omit the t, t' subscripts when the particular concurrent actions are immaterial.

There is no analogous result to show that the definition of concurrency is complete: that it includes every pair of coinitial transitions for which a cofinal notion of residuation might be defined. It is not entirely clear what form such a theorem might take, nor are we aware of any such theorem in the literature. Choice in particular is potentially problematic. Although by our (and Boudol and Castellani's) definition of $\smile$ coinitial choices are never concurrent, in the following we have distinct coinitial choices with "obvious" residuals, which are indeed cofinal:

$$\begin{tikzcd}
 & \underline{x}.P \arrow[dr, "\underline{x}.P"] & \\
(\underline{x}.P) + \underline{x}.P \arrow[ur, "\cdot + \underline{x}.P"] \arrow[dr, "(\underline{x}.P) + \cdot"'] & & P \\
 & \underline{x}.P \arrow[ur, "\underline{x}.P"'] &
\end{tikzcd}$$

No doubt more complex examples are possible. We leave exploring canonical notions of concurrency to future work.

3.3. *Examples of braiding*

Example 3 (Free braid).

Free braids arise when there are concurrent bound actions. For example the push injections used in the propagation rules $\cdot \overset{\underline{x}}{|}\, Q$ and $P \,|^{\underline{x}} \cdot$ open the process term with respect to index 0; if two of these happen consecutively, the order in which they happen determines the roles of indices 0 and 1 in the final process.

Concurrent name extrusions are analogous. In the process term $\nu\nu((\overline{x + 2}\langle 0 \rangle.P) \;|$

$\overline{z+2}\langle 1\rangle.Q)$ there are two binders that can be extruded; call the outer one ν_1 and the inner one ν_2. The output on the left extrudes ν_2, and the output on the right extrudes ν_1. Let t be the transition that extrudes ν_2:

$$\nu^{\overline{x}}\cdot\;\cfrac{\overline{\nu}\cdot\;\cfrac{\cdot^{\overline{x+2}\langle 0\rangle} \mid \overline{z+2}\langle 1\rangle.Q\;\cfrac{\Gamma+2\vdash \overline{x+2}\langle 0\rangle.P \xrightarrow{\overline{x+2}\langle 0\rangle} \Gamma+2\vdash P}{\Gamma+2\vdash(\overline{x+2}\langle 0\rangle.P)\mid\overline{z+2}\langle 1\rangle.Q \xrightarrow{\overline{x+2}\langle 0\rangle} \Gamma+2\vdash P\mid\overline{z+2}\langle 1\rangle.Q}}{\Gamma+1\vdash\nu((\overline{x+2}\langle 0\rangle.P)\mid\overline{z+2}\langle 1\rangle.Q) \xrightarrow{\overline{x+1}} \Gamma+2\vdash P\mid\overline{z+2}\langle 1\rangle.Q}}{\Gamma\vdash\nu\nu((\overline{x+2}\langle 0\rangle.P)\mid\overline{z+2}\langle 1\rangle.Q) \xrightarrow{\overline{x}} \Gamma+1\vdash\nu(\mathsf{swap}^{*}P\mid\overline{z+2}\langle 0\rangle.\mathsf{swap}^{*}Q)}$$

Here ν_2 is extruded as the bound output $\overline{x+1}$, propagating through the outer binder ν_1 as $\overline{x}$. In $\mathsf{tgt}(t)$, index 0 refers to the extruding ν_2; the binder remaining in the process term is ν_1. The key detail here is that the rule $\nu^{\overline{x}}\cdot$ moves ν_1 past ν_2, explaining the use of swap in $\mathsf{tgt}(t)$: whenever a propagating binder moves past a static binder, a swap must be applied under the static binder to preserve the local meaning of indices 0 and 1 (§ 2.2 above). This is also why $\overline{z+2}\langle 1\rangle$ in $\mathsf{src}(t)$ becomes $\overline{z+2}\langle 0\rangle$ in $\mathsf{tgt}(t)$.

Now let t' be the transition that extrudes ν_1:

$$\overline{\nu}\cdot\;\cfrac{\nu^{\overline{z+1}\langle 0\rangle}\cdot\;\cfrac{(\overline{x+2}\langle 0\rangle.P)\mid^{\overline{z+2}\langle 1\rangle}\cdot\;\cfrac{\Gamma+2\vdash\overline{z+2}\langle 1\rangle.Q \xrightarrow{\overline{z+2}\langle 1\rangle} \Gamma+2\vdash Q}{\Gamma+2\vdash(\overline{x+2}\langle 0\rangle.P)\mid\overline{z+2}\langle 1\rangle.Q \xrightarrow{\overline{z+2}\langle 1\rangle} \Gamma+2\vdash(\overline{x+2}\langle 0\rangle.P)\mid Q}}{\Gamma+1\vdash\nu((\overline{x+2}\langle 0\rangle.P)\mid\overline{z+2}\langle 1\rangle.Q) \xrightarrow{\overline{z+1}\langle 0\rangle} \Gamma+1\vdash\nu((\overline{x+2}\langle 0\rangle.P)\mid Q)}}{\Gamma\vdash\nu\nu((\overline{x+2}\langle 0\rangle.P)\mid\overline{z+2}\langle 1\rangle.Q) \xrightarrow{\overline{z}} \Gamma+1\vdash\nu((\overline{x+2}\langle 0\rangle.P)\mid Q)}$$

In this case, the output $\overline{x+2}\langle 1\rangle$ propagates through the inner binder ν_2 as $\overline{z+1}\langle 0\rangle$, and then becomes the extrusion $\overline{z}$ of the outer binder ν_1. In $\mathsf{tgt}(t')$, index 0 thus refers to the extruding ν_1, and the binder that remains in the process term is ν_2. The key detail here is that there is no swap in $\mathsf{tgt}(t')$ because this time the relative positions of ν_1 and ν_2 are unchanged: the extruding ν_1 is still the outer of the two binders.

The proof that t and t' are concurrent is straightforward because the outputs occur under opposite sides of a parallel composition. The notion of cofinality, however, is complicated by the use of indices to refer to ν_1 and ν_2. Naively, our expectation would be to derive t'/t and t/t' that complete the square

$$\begin{tikzcd}
& \Gamma+1\vdash\nu(\mathsf{swap}^{*}P\mid\overline{z+2}\langle 0\rangle.\mathsf{swap}^{*}Q) \arrow[dr, dotted, "t'/t"] & \\
\Gamma\vdash\nu\nu((\overline{x+2}\langle 0\rangle.P)\mid\overline{z+2}\langle 1\rangle.Q) \arrow[ur, "t"] \arrow[dr, "t'"'] & & \Gamma+\Delta\vdash R \\
& \Gamma+1\vdash\nu((\overline{x+2}\langle 0\rangle.P)\mid Q) \arrow[ur, dotted, "t/t'"'] &
\end{tikzcd}$$

for some $\Delta\in\{1,2\}$ and some process R. However, the only candidate for t'/t is to select the output redex on the right, which becomes an extrusion of the remaining binder, in this case ν_1:

$$\overline{\nu}\cdot\ \dfrac{\mathsf{swap}^*P \mid^{\overline{z+2}\langle 0\rangle} \cdot\ \dfrac{\Gamma+2\vdash \overline{z+2}\langle 0\rangle.\mathsf{swap}^*Q \xrightarrow{\overline{z+2}\langle 0\rangle} \Gamma+2\vdash \mathsf{swap}^*Q}{\Gamma+2\vdash \mathsf{swap}^*P \mid \overline{z+2}\langle 0\rangle.\mathsf{swap}^*Q \xrightarrow{\overline{z+2}\langle 0\rangle} \Gamma+2\vdash \mathsf{swap}^*P \mid \mathsf{swap}^*Q}}{\Gamma+1\vdash \nu(\mathsf{swap}^*P \mid \overline{z+2}\langle 0\rangle.\mathsf{swap}^*Q) \xrightarrow{\overline{z+2}} \Gamma+2\vdash \mathsf{swap}^*P \mid \mathsf{swap}^*Q}$$

leaving indices $0, 1$ referring to ν_1, ν_2 respectively in $\mathsf{tgt}(t'/t)$. Equally, the only candidate for the other residual t/t' is to select the output redex on the left, which also becomes an extrusion of the remaining binder, in this case ν_2:

$$\overline{\nu}\cdot\ \dfrac{\cdot\,^{\overline{x+2}\langle 0\rangle}\!\mid Q\ \ \dfrac{\Gamma+2\vdash \overline{x+2}\langle 0\rangle.P \xrightarrow{\overline{x+2}\langle 0\rangle} \Gamma+2\vdash P}{\Gamma+2\vdash (\overline{x+2}\langle 0\rangle.P) \mid Q \xrightarrow{\overline{x+2}\langle 0\rangle} \Gamma+2\vdash P \mid Q}}{\Gamma+1\vdash \nu((\overline{x+2}\langle 0\rangle.P) \mid Q) \xrightarrow{\overline{x+2}} \Gamma+2\vdash P \mid Q}$$

leaving indices $0, 1$ in $\mathsf{tgt}(t/t')$ referring to ν_2, ν_1 rather than ν_1, ν_2. So instead of the expected square, we have the pentagon

$$\begin{tikzcd}
 & \Gamma+1\vdash \nu(\mathsf{swap}^*P \mid \overline{z+2}\langle 0\rangle.\mathsf{swap}^*Q) \arrow[r, dotted, "t'/t"] & \Gamma+2\vdash \mathsf{swap}^*P \mid \mathsf{swap}^*Q \arrow[dd, dash, "\mathsf{swap}^*"] \\
\Gamma\vdash \nu\nu((\overline{x+2}\langle 0\rangle.P) \mid \overline{z+2}\langle 1\rangle.Q) \arrow[ur, "t"] \arrow[dr, "t'"'] & & \\
 & \Gamma+1\vdash \nu((\overline{x+2}\langle 0\rangle.P) \mid Q) \arrow[r, dotted, "t/t'"'] & \Gamma+2\vdash P \mid Q
\end{tikzcd}$$

with a swap path between $\mathsf{tgt}(t'/t)$ and $\mathsf{tgt}(t/t')$ reflecting the reordering of the propagating binders ν_1 and ν_2. ■

Example 4 (Propagating free braid).

Free braids are preserved by enclosing transitions as long as the residual actions of those transitions are bound. In particular, if a free braid propagates through a ν-binder it remains a free braid. Suppose $t \smile t'$ where the residual actions are both bound, so that $\mathsf{tgt}(t'/t) \asymp \mathsf{tgt}(t/t')$:

$$\begin{tikzcd}
 & \Gamma+2\vdash R \arrow[r, dotted, "(t'/t)^{\underline{z+2}}"] & \Gamma+3\vdash P' \arrow[dd, dash, "\mathsf{swap}^*"] \\
\Gamma+1\vdash P \arrow[ur, "t^{\underline{x+1}}"] \arrow[dr, "t'^{\underline{z+1}}"'] & & \\
 & \Gamma+2\vdash R' \arrow[r, dotted, "(t/t')^{\underline{x+2}}"'] & \Gamma+3\vdash \mathsf{swap}^*P'
\end{tikzcd}$$

Since both $\underline{x+1}$ and $\underline{z+1}$ are of the form push^*b, we can use the $\nu^b\cdot$ rule to form the composite transitions $\nu^{\underline{x}}t$ and $\nu^{\underline{z}}t'$ which propagate the input actions of t and t' actions through a ν-binder:

$$\nu^{\underline{x}}\cdot\ \dfrac{t\ \dfrac{\vdots}{\Gamma+1\vdash P\xrightarrow{\underline{x+1}}\Gamma+2\vdash R}}{\Gamma\vdash\nu P\xrightarrow{\underline{x}}\Gamma+1\vdash\nu(\mathsf{swap}^*R)}\qquad\qquad \nu^{\underline{z}}\cdot\ \dfrac{t'\ \dfrac{\vdots}{\Gamma+1\vdash P\xrightarrow{\underline{z+1}}\Gamma+2\vdash R'}}{\Gamma\vdash\nu P\xrightarrow{\underline{z}}\Gamma+1\vdash\nu(\mathsf{swap}^*R')}$$

Since $t\smile t'$ we can conclude $\nu^{\underline{x}}t\smile\nu^{\underline{z}}t'$ by the rules in Figure 7 and compute the following composite residual $(\nu^{\underline{z}}t')/\nu^{\underline{x}}t$:

$$\nu^{\underline{z+1}}\cdot\ \dfrac{\mathsf{swap}^*\cdot\ \dfrac{t'/t\ \dfrac{\vdots}{\Gamma+2\vdash R\xrightarrow{\underline{z+2}}\Gamma+3\vdash P'}}{\Gamma+2\vdash\mathsf{swap}^*R\xrightarrow{\underline{z+2}}\Gamma+3\vdash(\mathsf{swap}+1)^*P'}}{\Gamma+1\vdash\nu(\mathsf{swap}^*R)\xrightarrow{\underline{z+1}}\Gamma+2\vdash\nu(\mathsf{swap}^*(\mathsf{swap}+1)^*P')}$$

noting that $\mathsf{swap}^*(\underline{z+2})=\underline{z+2}$ by Lemma 8. The other residual $(\nu^{\underline{x}}t)/\nu^{\underline{z}}t'$ is similar but has an extra swap inherited from $\mathsf{tgt}(t/t')$:

$$\nu^{\overline{x+1}}\cdot\ \dfrac{\mathsf{swap}^*\cdot\ \dfrac{t/t'\ \dfrac{\vdots}{\Gamma+2\vdash R'\xrightarrow{\underline{x+2}}\Gamma+3\vdash\mathsf{swap}^*P'}}{\Gamma+2\vdash\mathsf{swap}^*R'\xrightarrow{\underline{x+2}}\Gamma+3\vdash(\mathsf{swap}+1)^*\mathsf{swap}^*P'}}{\Gamma+1\vdash\nu(\mathsf{swap}^*R')\xrightarrow{\underline{x+1}}\Gamma+2\vdash\nu(\mathsf{swap}^*(\mathsf{swap}+1)^*\mathsf{swap}^*P')}$$

Nevertheless, the target states of the composite residuals are still equated by swap, consistent with the fact that the residual actions still bound.

$$\begin{tikzcd}
 & \Gamma+1\vdash\nu(\mathsf{swap}^*R) \arrow[r, dotted, "\nu^{\underline{z+1}}(\mathsf{swap}^*t'/t)"] & \Gamma+2\vdash\nu(\mathsf{swap}^*(\mathsf{swap}+1)^*P') \arrow[d, dash, "\mathsf{swap}^*"] \\
\Gamma\vdash\nu P \arrow[ur, "\nu^{\underline{x}}t"] \arrow[dr, "\nu^{\underline{z}}t'"'] & & \Gamma+2\vdash\nu((\mathsf{swap}+1)^*\mathsf{swap}^*(\mathsf{swap}+1)^*P') \arrow[d, equal, "\nu\alpha"] \\
 & \Gamma+1\vdash\nu(\mathsf{swap}^*R') \arrow[r, dotted, "\nu^{\underline{x+1}}(\mathsf{swap}^*t/t')"'] & \Gamma+2\vdash\nu(\mathsf{swap}^*(\mathsf{swap}+1)^*\mathsf{swap}^*P')
\end{tikzcd}$$

Here α is the hexagon equating two ways of transposing indices 0 and 2 (Lemma 3) which $\nu\alpha$ lifts via congruence to an equality between one target and the swap image of the other. Thus $\asymp$ remains the appropriate notion of cofinality. ■

Example 5 (Bound braid).
A bound braid arises when concurrent ν-synchronisations have residuals that also ν-synchronise, which requires the underlying extrusions to be distinct binders. The concurrent transitions $t\smile t'$ and $u\smile u'$ below can be composed into concurrent ν-synchronisations that have this property; u has an input $\underline{x}$ matching the bound output $\overline{x}$ of t, and u' has a bound output $\overline{z}$ matching the input $\underline{z}$ of t'. The extrusions $\overline{x}$ and $\overline{z}$ are clearly of distinct binders since they arise on opposite sides of a parallel composition.

$$\begin{tikzcd}
 & \Gamma+1\vdash R \arrow[r, dotted, "(t'/t)^{\underline{z+1}}"] & \Gamma+2\vdash P' \arrow[d, dash, "\mathsf{swap}^*"] \\
\Gamma\vdash P \arrow[ur, "t^{\overline{x}}"] \arrow[dr, "t'^{\underline{z}}"'] & & {} \\
 & \Gamma+1\vdash R' \arrow[r, dotted, "(t/t')^{\overline{x+1}}"'] & \Gamma+2\vdash \mathsf{swap}^*P'
\end{tikzcd}
\qquad
\begin{tikzcd}
 & \Gamma+1\vdash S \arrow[r, dotted, "(u'/u)^{\overline{z+1}}"] & \Gamma+2\vdash Q' \arrow[d, dash, "\mathsf{swap}^*"] \\
\Gamma\vdash Q \arrow[ur, "u^{\underline{x}}"] \arrow[dr, "u'^{\overline{z}}"'] & & {} \\
 & \Gamma+1\vdash S' \arrow[r, dotted, "(u/u')^{\underline{x+1}}"'] & \Gamma+2\vdash \mathsf{swap}^*Q'
\end{tikzcd}$$

The composites are the ν-synchronisations $t\ {}_{\nu}{\mid}\ u : P \mid Q \xrightarrow{\tau} \nu(R \mid S)$ and $t' \mid_{\nu} u' : P \mid Q \xrightarrow{\tau} \nu(R' \mid S')$. Moreover since $t \smile t'$ and $u \smile u'$ we can conclude $t\ {}_{\nu}{\mid}\ u \smile t' \mid_{\nu} u'$ using the rules in Figure 6. The equations in Figure 7 determine the residual $(t' \mid_{\nu} u')/(t\ {}_{\nu}{\mid}\ u) = \nu^{\tau}(t'/t \mid_{\nu} u'/u)$, which we write down in full for clarity:

$$\nu^{\tau}\cdot\ \dfrac{\cdot \mid_{\nu} \cdot\ \dfrac{t'/t\ \dfrac{\vdots}{\Gamma+1\vdash S \xrightarrow{\underline{z+1}} \Gamma+2\vdash Q'} \qquad u'/u\ \dfrac{\vdots}{\Gamma+1\vdash R \xrightarrow{\overline{z+1}} \Gamma+2\vdash P'}}{\Gamma+1\vdash R \mid S \xrightarrow{\tau} \Gamma+1\vdash \nu(P' \mid Q')}}{\Gamma\vdash \nu(R \mid S) \xrightarrow{\tau} \Gamma\vdash \nu\nu(P' \mid Q')}$$

The other residual $(t' \mid_{\nu} u')/(t\ {}_{\nu}{\mid}\ u) = \nu^{\tau}(t'/t \mid_{\nu} u'/u)$ is similar, except it inherits two extra swap renamings from t/t' and u/u':

$$\nu^{\tau}\cdot\ \dfrac{\cdot\ {}_{\nu}{\mid}\ \cdot\ \dfrac{t/t'\ \dfrac{\vdots}{\Gamma+1\vdash S' \xrightarrow{\overline{x+1}} \Gamma+2\vdash \mathsf{swap}^*Q'} \qquad u/u'\ \dfrac{\vdots}{\Gamma+1\vdash R' \xrightarrow{\underline{x+1}} \Gamma+2\vdash \mathsf{swap}^*P'}}{\Gamma+1\vdash R' \mid S' \xrightarrow{\tau} \Gamma+1\vdash \nu(\mathsf{swap}^*P' \mid \mathsf{swap}^*Q')}}{\Gamma\vdash \nu(R' \mid S') \xrightarrow{\tau} \Gamma\vdash \nu\nu(\mathsf{swap}^*P' \mid \mathsf{swap}^*Q')}$$

Thus each residual ν-synchronises, and then propagates through the binder reinserted by the first synchronisation, leaving a double-ν in the final process. The residuals are related by the pentagon

$$\begin{tikzcd}
 & \Gamma\vdash \nu(R \mid S) \arrow[r, dotted, "\nu^{\tau}(t'/t \mid_{\nu} u'/u)"] & \Gamma\vdash \nu\nu(P' \mid Q') \arrow[d, dash, "\nu\nu\text{-}\mathsf{swap}_{P'\mid Q'}"] \\
\Gamma\vdash P \mid Q \arrow[ur, "t\ {}_{\nu}{\mid}\ u"] \arrow[dr, "t' \mid_{\nu} u'"'] & & {} \\
 & \Gamma\vdash \nu(R' \mid S') \arrow[r, dotted, "\nu^{\tau}(t/t'\ {}_{\nu}{\mid}\ u/u')"'] & \Gamma\vdash \nu\nu(\mathsf{swap}^*P' \mid \mathsf{swap}^*Q')
\end{tikzcd}$$

where $\nu\nu\text{-}\mathsf{swap}_{P'\mid Q'}$ is the bound braid that locates $P' \mid Q' \asymp \mathsf{swap}^*P' \mid \mathsf{swap}^*Q'$ under the two binders, representing the reordering of the binders. ■

Example 6 (Braid erasure by synchronisation). A free braid is erased if it is enclosed by a concurrent transition where the notion of cofinality is equality. For example, consider a variant of Example 5 where the extrusions $\overline{x}$ and $\overline{z}$ occur on the same side of the parallel composition, and represent extrusions of the same binder.

$$\begin{tikzcd}
& \Gamma + 1 \vdash R \arrow[r, dotted, "(t'/t)^{\underline{z+1}}"] & \Gamma + 2 \vdash P' \arrow[dd, dash, "\mathsf{swap}^*"] \\
\Gamma \vdash P \arrow[ur, "t^{\underline{x}}"] \arrow[dr, "t'^{\underline{z}}"'] & & \\
& \Gamma + 1 \vdash R' \arrow[r, dotted, "(t/t')^{\underline{x+1}}"'] & \Gamma + 2 \vdash \mathsf{swap}^* P'
\end{tikzcd}
\qquad
\begin{tikzcd}
& \Gamma + 1 \vdash S \arrow[r, dotted, "(u'/u)^{\overline{z+1}\langle 0 \rangle}"] & \Gamma + 1 \vdash Q' \arrow[dd, equal] \\
\Gamma \vdash Q \arrow[ur, "u^{\overline{x}}"] \arrow[dr, "u'^{\overline{z}}"'] & & \\
& \Gamma + 1 \vdash S' \arrow[r, dotted, "(u/u')^{\overline{x+1}\langle 0 \rangle}"'] & \Gamma + 1 \vdash Q'
\end{tikzcd}$$

The residuals u'/u and u/u' are plain outputs, rather than bound outputs. While the composites $t \mid_\nu u \smile t' \mid_\nu u'$ are concurrent ν-synchronisations as before, the residuals of the composites are plain synchronisations, again propagated through the ν-binder reinserted by the preceding step.

$$\begin{tikzcd}
& \Gamma \vdash \nu(R \mid S) \arrow[r, dotted, "\nu^\tau(t'/t \mid_0 u'/u)"] & \Gamma \vdash \nu((\mathsf{pop}\ 0)^* P' \mid Q') \arrow[dd, equal, "\nu(\alpha^* P'_= \mid Q'_=)"] \\
\Gamma \vdash P \mid Q \arrow[ur, "t \mid_\nu u"] \arrow[dr, "t' \mid_\nu u'"'] & & \\
& \Gamma \vdash \nu(R' \mid S') \arrow[r, dotted, "\nu^\tau(t/t' \mid_0 u/u')"'] & \Gamma \vdash \nu((\mathsf{pop}\ 0)^* \mathsf{swap}^* P' \mid Q')
\end{tikzcd}$$

Since the residual actions are plain τ actions, cofinality is simply equality. And indeed the substitution $\mathsf{pop}\ 0$ erases the free braid relating P' and $\mathsf{swap}^* P'$, by mapping indices 0 and 1 both to 0. Here α is the equality $(\mathsf{pop}\ 0) \circ \mathsf{swap} = \mathsf{pop}\ 0$ (Lemma 4) and $\nu(\alpha^* P' \mid Q'_=)$ uses congruence to lift α to an equivalence on target states, where $P'_=$ and $Q'_=$ denote their own canonical reflexivity proofs. ■

This completes our formal treatment of concurrent transitions in π-calculus, including the counterpart of the diamond lemma. In our setting, transitions may open terms with respect to variables, leading to a non-trivial notion of cofinality when such transitions are reordered. Like Boudol and Castellani, we omit a formalisation of Lévy's "cube" property, which extends the notion of concurrency to dimensions greater than two, since it is not required for the formalisation of causal equivalence.

4. Causal equivalence

We now turn to formalising *causal equivalence*, the congruence over sequences of transitions, or *traces*, induced by the concurrency relation for transitions. This is a standard concept from the theory of concurrent alphabets [Mazurkiewicz 1987], but is non-trivial in our setting because of braidings, which both propagate horizontally and compose vertically.

An "atom" of causal equivalence equates $t \cdot t'/t$ and $t' \cdot t/t'$ for concurrent transitions $t \smile t'$, where $t \cdot u$ denotes the composition of t and u. When the associated pentagon is composed horizontally into a larger computation, the continuation must be transported through the braiding $\gamma_{t,t'}$ which relates the target states of t'/t and t/t'. This requires two dimensions of closure, as illustrated in Figure 10. For coinitial u and $\gamma_{t,t'}$, the transition u must have an image $u/\gamma_{t,t'}$ in $\gamma_{t,t'}$, and the braiding $\gamma_{t,t'}$ must propagate as $\gamma_{t,t'}/u$:

$$\begin{tikzcd}
& Q \arrow[r, "t'/t"] & R \arrow[r, "u"] \arrow[dd, dash, "\gamma_{t,t'}"'] & S \arrow[dd, dash, dotted, "\gamma_{t,t'}/u"] \\
P \arrow[ur, "t"] \arrow[dr, "t'"'] & & & \\
& Q' \arrow[r, "t/t'"'] & R' \arrow[r, dotted, "u/\gamma_{t,t'}"'] & S'
\end{tikzcd}$$

Fig. 10: Closure of transitions under braidings

For braidings to be preserved by transitions in this way requires two generalisations to Definition 6. For free braids, we need the renaming to be of the form $\mathsf{swap}+\Delta$ rather than swap so that braid can shift under binders. For bound braids, we require the relation to be closed under reflexivity and parallel composition so that a braid can be dropped or duplicated via choice or replication. The earlier definition (Definition 3) placed a bound braid at a unique location in the process term.

An additional requirement is that braidings compose vertically when causal equivalences are composed via transitivity:

$$\begin{tikzcd}
& & & R \arrow[r, "u'/u"] & S \arrow[dd, dash, "\gamma_{u,u'}"] \\
& Q \arrow[r, "t'/t"] & P^{\dagger} \arrow[ur, "u"] \arrow[dr, "u'"'] \arrow[dd, dash, "\gamma_{t,t'}"'] & & \\
P \arrow[ur, "t"] \arrow[dr, "t'"'] & & & R' \arrow[r, "u/u'"] \arrow[dd, dash, dotted, "\gamma_{t,t'}/u'"] & S' \arrow[dd, dash, dotted, "(\gamma_{t,t'}/u')/(u/u')"] \\
& Q' \arrow[r, "t/t'"'] & P^{\ddagger} \arrow[dr, dotted, "u'/\gamma_{t,t'}"'] & & \\
& & & R'' \arrow[r, dotted, "(u/u')/(\gamma_{t,t'}/u')"'] & S''
\end{tikzcd}$$

Fig. 11: Sequential composition of concurrent transitions

This diagram represents the causal equivalence

$$t \cdot t'/t \cdot u \cdot u'/u \simeq t' \cdot t/t' \cdot u'/\gamma_{t,t'} \cdot (u/u')/(\gamma_{t,t'}/u')$$

with the targets S and S'' related by the composite braiding $\gamma_{u,u'} \cdot \gamma_{t,t'}/(\gamma_{t,t'}/u')/(u/u')$.

We proceed by defining traces $\boldsymbol{t}$ (§ 4.1), and then showing that, suitably generalised, braidings γ "commute" with coinitial traces $\boldsymbol{t}$, giving rise to residuals $\boldsymbol{t}/\boldsymbol{\gamma}$ and $\boldsymbol{\gamma}/\boldsymbol{t}$ (§ 4.2). These are used to define causal equivalences $\alpha : \boldsymbol{t} \simeq \boldsymbol{u}$ and composite braidings γ_α relating $\mathsf{tgt}(\boldsymbol{t})$ and $\mathsf{tgt}(\boldsymbol{u})$ (§ 4.3).

4.1. *Traces*

Define $\boldsymbol{a} : \mathsf{Action}^* \ \Gamma$ (bold $\boldsymbol{a}$) to be a finite sequence of composable actions starting at Γ, where a and a' are composable iff $a \in \mathsf{Action}\ \Gamma$ and $a' \in \mathsf{Action}\ (\Gamma + \mathsf{tgt}(a))$. $|\boldsymbol{a}|$ denotes the sum of $|a|$ for every a in $\boldsymbol{a}$. The empty sequence (nil) at Γ is written ε_Γ; extension to the left (cons) is written $a \cdot \boldsymbol{a}$. A *trace* $\boldsymbol{t} : P \xrightarrow{\boldsymbol{a}} R$ (bold $\boldsymbol{t}$) is a finite sequence of composable transitions, where t and u are composable iff $\mathsf{src}(u) = \mathsf{tgt}(t)$. The nil trace at P is written ε_P; cons of $t : P \xrightarrow{a} R$ onto $\boldsymbol{t} : R \xrightarrow{\boldsymbol{a}} S$ is written $t \cdot \boldsymbol{t} : P \xrightarrow{a \cdot \boldsymbol{a}} S$.

The renamings $\rho^* a$ and $\rho^* t$ of an action and a transition extend to action sequences and traces respectively.

Lemma 11 (Lifting of renamings to action sequences and traces).
Suppose $\rho : \Gamma \longrightarrow \Delta$ and $\boldsymbol{t} : P \xrightarrow{\boldsymbol{a}} R$, where $\Gamma \vdash P$, $\Gamma + \Gamma' \vdash R$ and $\boldsymbol{a} : \mathsf{Action}^* \ \Gamma$.

$$\begin{tikzcd}
\Gamma \vdash P \arrow[r, "\boldsymbol{t}^{\boldsymbol{a}}"] \arrow[d, "\rho^*"'] & \Gamma + \Gamma' \vdash R \arrow[d, "(\rho + \Gamma')^*"] \\
\Delta \vdash \rho^* P \arrow[r, dotted, "(\rho^* \boldsymbol{t})^{\rho^* \boldsymbol{a}}"'] & \Delta + \Gamma' \vdash (\rho + \Gamma')^* R
\end{tikzcd}$$

Then there exist actions $\rho^* \boldsymbol{a} : \mathsf{Action}^* \ \Delta$ and trace $\rho^* \boldsymbol{t} : \rho^* P \xrightarrow{\rho^* \boldsymbol{a}} (\rho + \Gamma')^* R$.

Proof. By the following defining equations.

$$\begin{array}{ll}
\rho^* \varepsilon_\Gamma = \varepsilon_\Delta & \rho^*(b \cdot \boldsymbol{a}) = (\rho^* b) \cdot (\rho + 1)^* \boldsymbol{a} \\
 & \rho^*(c \cdot \boldsymbol{a}) = (\rho^* c) \cdot \rho^* \boldsymbol{a} \\
\rho^* \varepsilon_P = \varepsilon_P & \rho^*(t^b \cdot \boldsymbol{t}) = (\rho^* t^b) \cdot (\rho + 1)^* \boldsymbol{t} \\
 & \rho^*(t^c \cdot \boldsymbol{t}) = (\rho^* t^c) \cdot \rho^* \boldsymbol{t}
\end{array}$$

□

4.2. *Residuals of trace and braiding*

We now require a minimal generalisation of our system of transitions and braidings sufficient to admit the following notions of residuation:

$$\begin{tikzcd}
\Gamma \vdash P \arrow[r, "t"] \arrow[d, dash, "\gamma"'] & \Gamma + \Delta \vdash R \arrow[d, dotted, dash, "\gamma/t"] \\
\Gamma \vdash P' \arrow[r, dotted, "t/\gamma"'] & \Gamma + \Delta \vdash R'
\end{tikzcd}$$

where $\Delta \in \{0, 1\}$ and γ relates the target states of an earlier concurrent transition, as shown in Figure 10. Recall that γ witnesses either $P \mathrel{\underline{\times}} P'$, $P \mathrel{\overline{\times}} P'$ or $P = P'$ (Definition 6). We consider the residual γ/t in each case.

Case $P \mathrel{\underline{\times}} P'$. Then $P = \mathsf{swap}^* P'$ and $R = (\mathsf{swap} + \Delta)^* R'$ by Lemma 9. If $\Delta = 1$ then the braid has shifted under a binder and thus $R \not\mathrel{\underline{\times}} R'$. Therefore the first generalisation closes free braids under translations by an arbitrary Δ. We define the following relation, noting that $\mathrel{\underline{\times}} = \mathrel{\underline{\times}}_0$.

Definition 5 (Free braid, generalised). For any processes $\Gamma + 2 + \Delta \vdash P, R$ define the symmetric relation $P \underline{\times}_{\Delta} R$ as follows. The context Γ is left implicit.

$$P \underline{\times}_{\Delta} R \quad\Leftrightarrow\quad P = (\mathsf{swap}_{\Gamma} + \Delta)^{*} R$$

Case $P \overline{\times} P'$. Whereas a free braid inserts a swap renaming at the root of P, a bound braid inserts a swap under exactly one pair of adjacent binders in P. When a transition $t : P \xrightarrow{a} R$ is taken, subterms of P may be dropped or duplicated: in particular non-taken branches of choices are discarded, and the bodies of replications are copied into both sides of the resulting parallel compositions. It may therefore not be possible to obtain R' from R by inserting exactly one bound swap, in which case $R \not\overline{\times} R'$. The second generalisation thus closes bound braids under reflexivity (for dropping) and parallel composition (for duplication). Figure 12 defines the new relation, also written $\overline{\times}$.

$$\boxed{P \overline{\times} R}$$

$$\nu\nu\text{-}\mathsf{swap}_P\ \frac{P \underline{\times} P'}{\nu\nu P \overline{\times} \nu\nu P'} \qquad \mathbf{0}\ \frac{}{\mathbf{0} \overline{\times} \mathbf{0}} \qquad \underline{x}.P\ \frac{}{\underline{x}.P \overline{\times} \underline{x}.P} \qquad \overline{x}\langle y\rangle.P\ \frac{}{\overline{x}\langle y\rangle.P \overline{\times} \overline{x}\langle y\rangle.P}$$

$$\cdot + Q\ \frac{P \overline{\times} R}{P + Q \overline{\times} R + Q} \qquad P + \cdot\ \frac{Q \overline{\times} S}{P + Q \overline{\times} P + S} \qquad \cdot \mid \cdot\ \frac{P \overline{\times} R \qquad Q \overline{\times} S}{P \mid Q \overline{\times} R \mid S}$$

$$\nu\cdot\ \frac{P \overline{\times} R}{\nu P \overline{\times} \nu R} \qquad !\cdot\ \frac{P \overline{\times} R}{!P \overline{\times} !R}$$

Fig. 12: Bound braid $P \overline{\times} R$ that can be dropped or duplicated

Case $P = P'$. The situation is trivial and γ/t is simply the reflexivity proof that $R = R'$.

The three cases above determine a new braiding relation $\overline{\underline{\times}}_{\mathring{a},\Delta}$ which is closed under transitions.

Definition 6 (Braiding, generalised). For any contexts Γ, Δ, any $a, a' \in \mathsf{Action}\ \Gamma$ and any $\mathring{a} : a \smile a'$, define the following symmetric relation $\overline{\underline{\times}}_{\mathring{a},\Delta}$ over processes in $\Gamma' + \Delta$, where Γ' is the target context of $\mathring{a}$. There are only two cases rather than three, since the $=$ case is now subsumed by the reflexivity of bound braids.

$$\overline{\underline{\times}}_{\mathring{a},\Delta} \overset{\text{def}}{=} \begin{cases} \underline{\times}_{\Delta} & \text{if } \overline{\underline{\times}}_{\mathring{a}} = \underline{\times} \\ \overline{\times} & \text{otherwise} \end{cases}$$

Since $\underline{\times} = \underline{\times}_0$, and there is an obvious embedding, via reflexivity, of the old definition of $\overline{\times}$ (Figure 9) into the new one, there is also an embedding of $\overline{\underline{\times}}_{\mathring{a}}$ into $\overline{\underline{\times}}_{\mathring{a},0}$.

Lemma 12. $\overline{\underline{\times}}_{\mathring{a}} \subseteq \overline{\underline{\times}}_{\mathring{a},0}$

The new braidings are closed under transitions, so we can define the required residuals γ/t and t/γ. We start with the case when γ is a bound braid ϕ. Note that subsuming the

$=$ case into the reflexivity of $\overline{\times}$ does not lose any precision, since for any $t : P \xrightarrow{a} R$ we have $t/P_{\overline{\times}} = t$ and thus $P_{\overline{\times}}/t = R_{\overline{\times}}$.

Theorem 2. Suppose $t : P \xrightarrow{a} R$ and $\phi : P \,\overline{\times}\, P'$. Then there exists a process R', transition $t/\phi : P' \xrightarrow{a} R'$ and bound braid $\phi/t : R \,\overline{\times}\, R'$.

$$\begin{tikzcd}
P \arrow[r, "t"] \arrow[d, dash, "\phi"'] & R \arrow[d, dotted, dash, "\phi/t"] \\
P' \arrow[r, dotted, "t/\phi"'] & R'
\end{tikzcd}$$

Proof. By the defining equations in Figure 13. Unlike residuals of the form t/t', the cofinality of t/ϕ and ϕ/t is by construction. $P_{\overline{\times}}$ denotes the reflexivity proof that $P \,\overline{\times}\, P$.

$\boxed{t/\phi}$

$$\begin{aligned}
(\overline{\nu}\nu^{\overline{x+1}\langle 0\rangle} t)/\nu\nu\text{-}\mathsf{swap}_{\mathsf{src}(t)} &= \nu^{\overline{x}}\overline{\nu}(\mathsf{swap}^* t)\\
(\nu^{\overline{x}}\overline{\nu} t)/\nu\nu\text{-}\mathsf{swap}_{\mathsf{src}(t)} &= \overline{\nu}\nu^{\overline{x+1}\langle 0\rangle}(\mathsf{swap}^* t)\\
(\nu^{c}\nu^{c'} t)/\nu\nu\text{-}\mathsf{swap}_{\mathsf{src}(t)} &= \nu^{c}\nu^{c'}(\mathsf{swap}^* t)\\
(\nu^{b}\nu^{b'} t)/\nu\nu\text{-}\mathsf{swap}_{\mathsf{src}(t)} &= \nu^{b}\nu^{b'}(\mathsf{swap}^* t)\\
(\underline{x}.P)/(\underline{x}.\phi) &= \underline{x}.\mathsf{tgt}(\phi)\\
(\overline{x}\langle y\rangle.P)/(\overline{x}\langle y\rangle.\phi) &= \overline{x}\langle y\rangle.\mathsf{tgt}(\phi)\\
(t+Q)/(\phi+Q) &= t/\phi + Q\\
(t+Q)/(P+\psi) &= t + \mathsf{tgt}(\psi)\\
(P+u)/(P+\psi) &= P + u/\psi\\
(P+u)/(\phi+Q) &= \mathsf{tgt}(\phi) + u\\
(t\ {}^{b}|\ Q)/(\phi \mid \psi) &= t/\phi\ {}^{b}|\ \mathsf{tgt}(\psi)\\
(t\ {}^{c}|\ Q)/(\phi \mid \psi) &= t/\phi\ {}^{c}|\ \mathsf{tgt}(\psi)\\
(P\ |^{b}\ u)/(\phi \mid \psi) &= \mathsf{tgt}(\phi)\ |^{b}\ u/\psi\\
(P\ |^{c}\ u)/(\phi \mid \psi) &= \mathsf{tgt}(\phi)\ |^{c}\ u/\psi\\
(t\ |_{y}\ u)/(\phi \mid \psi) &= t/\phi\ |_{y}\ u/\psi\\
(t\ {}_{y}|\ u)/(\phi \mid \psi) &= t/\phi\ {}_{y}|\ u/\psi\\
(t\ |_{\nu}\ u)/(\phi \mid \psi) &= t/\phi\ |_{\nu}\ u/\psi\\
(t\ {}_{\nu}|\ u)/(\phi \mid \psi) &= t/\phi\ {}_{\nu}|\ u/\psi\\
(\overline{\nu} t)/(\nu\phi) &= \overline{\nu}\ t/\phi\\
(\nu^{b} t)/(\nu\phi) &= \nu^{b}\, t/\phi\\
(\nu^{c} t)/(\nu\phi) &= \nu^{c}\, t/\phi\\
(!t)/(!\phi) &= !t/(\phi \mid !\phi)
\end{aligned}$$

$\boxed{\phi/t}$

$$\begin{aligned}
\nu\nu\text{-}\mathsf{swap}_{\mathsf{src}(t)}/(\overline{\nu}\nu^{\overline{x+1}\langle 0\rangle} t) &= \nu\ \mathsf{tgt}(t)_{\overline{\times}}\\
\nu\nu\text{-}\mathsf{swap}_{\mathsf{src}(t)}/(\nu^{\overline{x}}\overline{\nu} t) &= \nu\ (\mathsf{swap}^*\mathsf{tgt}(t))_{\overline{\times}}\\
\nu\nu\text{-}\mathsf{swap}_{\mathsf{src}(t)}/(\nu^{c}\nu^{c'} t) &= \nu\nu\text{-}\mathsf{swap}_{\mathsf{tgt}(t)}\\
\nu\nu\text{-}\mathsf{swap}_{\mathsf{src}(t)}/(\nu^{b}\nu^{b'} t) &= \nu\nu\text{-}\mathsf{swap}_{\mathsf{swap}^*(\mathsf{swap}+1)^*\mathsf{swap}^*\mathsf{tgt}(t)}\\
(\underline{x}.\phi)/(\underline{x}.P) &= \phi\\
(\overline{x}\langle y\rangle.\phi)/(\overline{x}\langle y\rangle.P) &= \phi\\
(\phi+Q)/(t+Q) &= \phi/t\\
(P+\psi)/(t+Q) &= \mathsf{tgt}(t)_{\overline{\times}}\\
(P+\psi)/(P+u) &= \psi/u\\
(\phi+Q)/(P+u) &= \mathsf{tgt}(u)_{\overline{\times}}\\
(\phi \mid \psi)/(t\ {}^{b}|\ Q) &= \phi/t \mid \mathsf{push}^*\psi\\
(\phi \mid \psi)/(t\ {}^{c}|\ Q) &= \phi/t \mid \psi\\
(\phi \mid \psi)/(P\ |^{b}\ u) &= \mathsf{push}^*\phi \mid \psi/u\\
(\phi \mid \psi)/(P\ |^{c}\ u) &= \phi \mid \psi/u\\
(\phi \mid \psi)/(t\ |_{y}\ u) &= (\mathsf{pop}\ y)^*\phi/t \mid \psi/u\\
(\phi \mid \psi)/(t\ {}_{y}|\ u) &= \phi/t \mid (\mathsf{pop}\ y)^*\psi/u\\
(\phi \mid \psi)/(t\ |_{\nu}\ u) &= \nu(\phi/t \mid \psi/u)\\
(\phi \mid \psi)/(t\ {}_{\nu}|\ u) &= \nu(\phi/t \mid \psi/u)\\
(\nu\phi)/(\overline{\nu} t) &= \phi/t\\
(\nu\phi)/(\nu^{b} t) &= \nu\ \mathsf{swap}^*\phi/t\\
(\nu\phi)/(\nu^{c} t) &= \nu\ \phi/t\\
(!\phi)/(!t) &= (\phi \mid !\phi)/t
\end{aligned}$$

Fig. 13: Residuals of transition t and coinitial bound braid ϕ

Figure 14 illustrates Theorem 2 for the cases where ϕ is of the form $\nu\nu\text{-}\mathsf{swap}_P$, omitting the various renaming lemmas used as type-level coercions.

$$\begin{tikzcd}
\Gamma \vdash \nu\nu(\mathsf{swap}^* P) \arrow[r, "\overline{\nu}\nu^{\overline{x+1}\langle 0\rangle} t"] \arrow[d, dash, "\nu\nu\text{-}\mathsf{swap}_P"'] & \Gamma + 1 \vdash \nu R \arrow[d, equal] \\
\Gamma \vdash \nu\nu P \arrow[r, dotted, "\nu^{\overline{x}}\overline{\nu}(\mathsf{swap}^* t)"'] & \Gamma + 1 \vdash \nu R
\end{tikzcd}
\qquad
\begin{tikzcd}
\Gamma \vdash \nu\nu(\mathsf{swap}^* P) \arrow[r, "\nu^{c}\nu^{c'} t"] \arrow[d, dash, "\nu\nu\text{-}\mathsf{swap}_P"'] & \Gamma \vdash \nu\nu R \arrow[d, dotted, dash, "\nu\nu\text{-}\mathsf{swap}_R"] \\
\Gamma \vdash \nu\nu P \arrow[r, dotted, "\nu^{c}\nu^{c'}(\mathsf{swap}^* t)"'] & \Gamma \vdash \nu\nu(\mathsf{swap}^* R)
\end{tikzcd}$$

$$\begin{tikzcd}
\Gamma \vdash \nu\nu(\mathsf{swap}^* P) \arrow[r, "\nu^{\overline{x}}\overline{\nu} t"] \arrow[d, dash, "\nu\nu\text{-}\mathsf{swap}_P"'] & \Gamma + 1 \vdash \nu(\mathsf{swap}^* R) \arrow[d, equal] \\
\Gamma \vdash \nu\nu P \arrow[r, dotted, "\overline{\nu}\nu^{\overline{x+1}\langle 0\rangle}(\mathsf{swap}^* t)"'] & \Gamma + 1 \vdash \nu(\mathsf{swap}^* R)
\end{tikzcd}
\qquad
\begin{tikzcd}
\Gamma \vdash \nu\nu(\mathsf{swap}^* P) \arrow[r, "\nu^{b}\nu^{b'} t"] \arrow[d, dash, "\nu\nu\text{-}\mathsf{swap}_P"'] & \Gamma + 1 \vdash \nu\nu((\mathsf{swap} + 1)^* \mathsf{swap}^* R) \arrow[d, dotted, dash, "\nu\nu\text{-}\mathsf{swap}_{(\mathsf{swap}+1)^* \mathsf{swap}^* R}"] \\
\Gamma \vdash \nu\nu P \arrow[r, dotted, "\nu^{b}\nu^{b'}(\mathsf{swap}^* t)"'] & \Gamma + 1 \vdash \nu\nu(\mathsf{swap}^*(\mathsf{swap} + 1)^* \mathsf{swap}^* R)
\end{tikzcd}$$

Fig. 14: Cofinality of ϕ/t and t/ϕ in the $\nu\nu$-swap cases

It is then straightforward to extend t/ϕ and ϕ/t to braidings γ and traces $\boldsymbol{t}$.

Lemma 13 (Residuals of transition t and γ). Suppose $t : P \xrightarrow{a} R$ and $\gamma : P \mathrel{\overline{\underline{\times}}}_{\mathring{a},\Delta} P'$. Then there exists process R', action a/γ, transition $t/\gamma : P' \xrightarrow{a/\gamma} R'$ and braiding $\gamma/t : R \mathrel{\overline{\underline{\times}}}_{\mathring{a},\Delta'} R'$, where $\Delta' = \Delta + |a|$.

$$\begin{tikzcd}
\Gamma + \Delta \vdash P \arrow[r, "t"] \arrow[d, dash, "\gamma"'] & \Gamma + \Delta' \vdash R \arrow[d, dotted, dash, "\gamma/t"] \\
\Gamma + \Delta \vdash P' \arrow[r, dotted, "t/\gamma"'] & \Gamma + \Delta' \vdash R'
\end{tikzcd}$$

Proof. By the following defining equations, which are given for t/γ and γ/t simultaneously. As usual $P_=$ denotes the reflexivity proof that $P = P$.

$$(t/\gamma, \gamma/t) = \begin{cases} ((\mathsf{swap} + \Delta)^* t, ((\mathsf{swap} + \Delta')^* R)_=) & \text{if } P \mathrel{\underline{\times}}_{\Delta} P' \\ (t/\phi, \phi/t) & \text{if } P \mathrel{\overline{\times}} P' \text{ and } \gamma = \phi \end{cases}$$

The diagram for Lemma 14 is the same as for Lemma 13 but with $\boldsymbol{t}$ instead of t.

Lemma 14 (Residuals of trace $\boldsymbol{t}$ and γ).
Suppose $\boldsymbol{t} : P \xrightarrow{\boldsymbol{a}} R$ and $\gamma : P \mathrel{\overline{\underline{\times}}}_{\mathring{a},\Delta} P'$. Then there exists process R', action sequence $\boldsymbol{a}/\gamma$, trace $\boldsymbol{t}/\gamma : P' \xrightarrow{\boldsymbol{a}/\gamma} R'$ and braiding $\gamma/\boldsymbol{t} : R \mathrel{\overline{\underline{\times}}}_{\mathring{a},\Delta'} R'$, where $\Delta' = \Delta + |\boldsymbol{a}|$.

Proof. By the following defining equations.

$$\begin{tikzcd}
P \arrow[r, equal, "\varepsilon_P"] \arrow[d, dash, "\boldsymbol{\gamma}"'] & P \arrow[d, dash, "\boldsymbol{\gamma}"] \\
P' \arrow[r, equal, "\varepsilon_{P'}"'] & P'
\end{tikzcd}
\qquad
\begin{tikzcd}
P \arrow[r, "t"] \arrow[d, dash, "\boldsymbol{\gamma}"'] & R \arrow[r, "\boldsymbol{t}"] \arrow[d, dotted, dash, "\boldsymbol{\gamma}/t"] & S \arrow[d, dotted, dash, "(\boldsymbol{\gamma}/t)/\boldsymbol{t}"] \\
P' \arrow[r, dotted, "t/\boldsymbol{\gamma}"'] & R' \arrow[r, dotted, "\boldsymbol{t}/(\boldsymbol{\gamma}/t)"'] & S'
\end{tikzcd}$$

$$\varepsilon_P/\boldsymbol{\gamma} = \varepsilon_{P'} \qquad (t \cdot \boldsymbol{t})/\boldsymbol{\gamma} = t/\boldsymbol{\gamma} \cdot \boldsymbol{t}/(\boldsymbol{\gamma}/t)$$
$$\boldsymbol{\gamma}/\varepsilon_P = \boldsymbol{\gamma} \qquad \boldsymbol{\gamma}/(t \cdot \boldsymbol{t}) = (\boldsymbol{\gamma}/t)/\boldsymbol{t}$$

4.3. *Causal equivalence*

A causal equivalence $\alpha : \boldsymbol{t} \simeq \boldsymbol{u}$ reorders a trace $\boldsymbol{t}$ into an equal-length, coinitial trace $\boldsymbol{u}$ by permuting concurrent transitions. Meta-variables α, β range over causal equivalences. If $\alpha : \boldsymbol{t} \simeq \boldsymbol{u}$ then $\mathsf{tgt}(t)$ and $\mathsf{tgt}(u)$ are related by a unique braiding $\boldsymbol{\gamma}_\alpha$.

In what follows, rules which mention a trace of the form $t \cdot \boldsymbol{t}$ have an implicit side-condition asserting $\mathsf{tgt}(t) = \mathsf{src}(\boldsymbol{t})$, and rules which mention a braiding $\boldsymbol{\gamma}_{t,t'}$ have an implicit side-condition asserting $t \smile t'$.

Definition 7. Inductively define the relation $\simeq$ using the rules in Figure 15, where syntactically $\simeq$ has lower priority than $\cdot$.

$$\boxed{\boldsymbol{t} \simeq \boldsymbol{u}}$$

$$\varepsilon_P \ \frac{}{\varepsilon_P \simeq \varepsilon_P} \qquad\qquad t \cdot \cdot \ \frac{\boldsymbol{t} \simeq \boldsymbol{u}}{t \cdot \boldsymbol{t} \simeq t \cdot \boldsymbol{u}}$$

$$(t \smile t') \cdot \boldsymbol{t} \ \frac{}{t \cdot t'/t \cdot \boldsymbol{t} \simeq t' \cdot t/t' \cdot \boldsymbol{t}/\boldsymbol{\gamma}_{t,t'}} \qquad\qquad \cdot \circ \cdot \ \frac{\boldsymbol{t}' \simeq \boldsymbol{u} \qquad \boldsymbol{t} \simeq \boldsymbol{t}'}{\boldsymbol{t} \simeq \boldsymbol{u}}$$

Fig. 15: Causal equivalence

The ε_P and $t \cdot \cdot$ rules are the congruence cases. The $\cdot \circ \cdot$ rule closes under transitivity, which is a form of vertical composition and which also causes braidings to compose vertically. The transposition rule $(t \smile t') \cdot \boldsymbol{t}$ composes a concurrent pair $t \smile t'$ with a continuation $\boldsymbol{t}$ for t'/t, transporting $\boldsymbol{t}$ through the braiding $\boldsymbol{\gamma}_{t,t'}$ witnessing the cofinality of t and t' to obtain the continuation $\boldsymbol{t}/\boldsymbol{\gamma}_{t,t'}$ for t/t', as shown in Figure 16.

$$\begin{tikzcd}
 & R \arrow[r, "t'/t"] & Q \arrow[r, "\boldsymbol{t}"] \arrow[dd, dash, "\boldsymbol{\gamma}_{t,t'}"'] & S \arrow[dd, dotted, dash, "\boldsymbol{\gamma}_{t,t'}/\boldsymbol{t}"] \\
P \arrow[ur, "t"] \arrow[dr, "t'"'] & & & \\
 & R' \arrow[r, "t/t'"'] & Q' \arrow[r, dotted, "\boldsymbol{t}/\boldsymbol{\gamma}_{t,t'}"'] & S'
\end{tikzcd}$$

Fig. 16: Causal equivalence, transposition rule

Theorem 3. $\simeq$ is an equivalence relation.

Proof. Reflexivity is a trivial induction, using the ε_P and $t \cdot \alpha$ rules. Transitivity is immediate from the $\alpha \circ \beta$ rule. Symmetry is trivial in the ε_P, $t \cdot \alpha$ and $\alpha \circ \beta$ cases. The $(t \smile t') \cdot \boldsymbol{t}$ case requires the symmetry of $\smile$ and that $(\boldsymbol{t}/\gamma)/\gamma = \boldsymbol{t}$. $\square$

A causal equivalence $\alpha : \boldsymbol{t} \simeq \boldsymbol{u}$ determines a composite braiding relation $⧖_\alpha$ which precisely sequences the atomic braidings required to relate $\mathsf{tgt}(\boldsymbol{t})$ to $\mathsf{tgt}(\boldsymbol{u})$.

Definition 8 (Braiding for equivalent traces). Inductively define the family of relations $⧖_\alpha$ for any $a : \boldsymbol{t} \simeq \boldsymbol{u}$, using the rules in Figure 17.

$$\boxed{P ⧖_\alpha R}$$

$$\frac{}{P ⧖_{\varepsilon_P} P} \qquad \frac{}{P ⧖_{(t \smile t') \cdot \boldsymbol{t}} R}\ \boldsymbol{\gamma}_{t,t'}/\boldsymbol{t} : P ⧖_{\mathring{a},\Delta} R \qquad \frac{P ⧖_\alpha R}{P ⧖_{t \cdot \alpha} R} \qquad \frac{P ⧖_\beta R \qquad R ⧖_\alpha S}{P ⧖_{\alpha \circ \beta} S}$$

Fig. 17: Braiding relation $⧖_\alpha$ relating $\mathsf{tgt}(t)$ and $\mathsf{tgt}(u)$ for any $\alpha : \boldsymbol{t} \simeq \boldsymbol{u}$

As with $⧖_{\mathring{a},\Delta}$, the relation $⧖_\alpha$ is a singleton, inhabited by a unique path $\boldsymbol{\gamma}_\alpha$ between $\mathsf{tgt}(\boldsymbol{t})$ and $\mathsf{tgt}(\boldsymbol{u})$. (However α itself is not unique, since there are many ways of proving $\boldsymbol{t} \simeq \boldsymbol{u}$.) The $P ⧖_{\varepsilon_P} P$ case is an empty composite braiding. The $P ⧖_{(t \smile t') \cdot \boldsymbol{t}} R$ case turns an atomic braiding $\boldsymbol{\gamma}_{t,t'}$ into one step of a composite braiding, after transporting it through the continuation $\boldsymbol{t}$. The $P ⧖_{t \cdot \alpha} R$ case simply recognises that $\mathsf{tgt}(t \cdot \boldsymbol{t}) = \mathsf{tgt}(\boldsymbol{t})$. Finally $P ⧖_{\alpha \circ \beta} R$ is the composition rule, closing under transitivity.

Theorem 4. Suppose $\alpha : \boldsymbol{t} \simeq \boldsymbol{u}$. Then there exists a unique $\boldsymbol{\gamma}_\alpha : \mathsf{tgt}(\boldsymbol{t}) ⧖_\alpha \mathsf{tgt}(\boldsymbol{u})$.

Theorem 5. $⧖_\alpha$ is a $\simeq$-indexed family of equivalence relations.

5. Related work

The μs calculus [Hirschkoff 1999] has a similar treatment of de Bruijn indices. Its renaming operators $\langle x \rangle$, ϕ and ψ are effectively our $\mathsf{pop}\ x$, push and swap renamings, but fused with the $\cdot^*$ operator which applies a renaming to a process. Hirschkoff's operators are also syntactic forms in the μs calculus, rather than meta-operations, and therefore the operational semantics also includes rules for reducing occurrences of the renaming operators that arise during a process reduction step.

As noted earlier in the paper, our approach to defining causal equivalence of traces is influenced by a line of work stemming from the study of optimal reduction in the λ-calculus [Lévy 1980], via the "proved transition" semantics of CCS [Boudol and Castellani 1989].

Boreale and Sangiorgi [1998] and Degano and Priami [1999] investigate causality in the context of the π-calculus. Similar ideas (from which we also drew inspiration) appear in work on reversible CCS, such as RCCS [Danos and Krivine 2004], and reversible π-calculi, such as $\rho\pi$ [Lanese et al. 2010] and Rπ [Cristescu et al. 2013]). Reversible calculi equip process terms with additional structure to support undoing actions; causal equivalence and permutation of transitions is necessary here to allow undoing actions in a different (sequential) order than they were performed. However, this additional structure changes the metatheory: for example, in Rπ two traces are coinitial and cofinal if and only if they are equivalent, which does not hold in our setting. To the best of our knowledge, there is no prior work that presents a proved transition semantics for a "vanilla" π-calculus, rather than an augmented variant.

Formalisations of the π-calculus have been undertaken in several theorem provers used for mechanised metatheory, including Coq, HOL, Isabelle/HOL, Nominal Isabelle, CLF, Abella, and Agda.

HOL Melham [1994] reports on a formalisation of the π-calculus in HOL, using names axiomatised as an unspecified, infinite set, and following Milner et al. [1992] closely. Substitution is parameterised over a choice function specifying how to choose a name fresh for a given set of names, which is used to rename bound names to avoid capture. Aït Mohamed [1995] formalised the π-calculus in HOL using concrete syntax and verified proof rules for early bisimulation checking.

Coq An early mechanisation of residuation theory was Huet's formalisation in Coq of residuals for λ-calculus [Huet 1994], which also uses de Bruijn indices. Huet's chief contribution is an inductive definition of residual, a proof that residuals commute with substitution, and a "prism" theorem that generalises Lévy's cube lemma.

Hirschkoff [1997] formalised the π-calculus in Coq using de Bruijn indices, and verified properties such as congruence and structural equivalence laws of bisimulation. Despeyroux [2000] formalised the π-calculus in Coq using weak higher-order abstract syntax, assuming a decidable type of names, and using two separate transitions, for ordinary, input and output transitions respectively; for input and output transitions the right-hand side is a function of type $\mathsf{name} \longrightarrow \mathsf{proc}$. This formalisation included a simple type system and proof of type soundness. Honsell et al. [2001] formalised the π-calculus in Coq, also using weak higher-order abstract syntax. The type of names name is a type parameter assumed to admit decidable equality and freshness (notin) relations. Transitions are encoded using two inductive definitions, for free and bound actions, which differ in the type of the third argument (proc vs. $\mathsf{name} \longrightarrow \mathsf{proc}$). Numerous results from Milner et al. [1992] are verified, using the *theory of contexts* (whose axioms are assumed in their formalisation, but have been validated semantically by Bucalo et al. [2006]).

Affeldt and Kobayashi [2008] developed a library based on a variant of the π-calculus (with channels typed using Coq types) for representing and reasoning about concurrent processes. Processes are represented using higher-order abstract syntax, and exotic terms are allowed; some lemmas are not formally proved but introduced as axioms with semantic justifications.

Isabelle/HOL Röckl et al. [2001] and Röckl and Hirschkoff [2003] formalised the π-calculus in Isabelle/HOL and verified properties such as adequacy, following the theory of contexts approach to higher-order abstract syntax introduced by Honsell et al. [2001], and using well-formedness predicates to rule out exotic terms. Gay [2001] developed a framework for formalising (linear) type systems for the π-calculus in Isabelle/HOL, using de Bruijn indices for binding syntax and a reduction-style semantics rather than labelled transitions.

Abella Tiu and Miller [2010] encode the syntax and semantics of the π-calculus using the λ-term abstract syntax variant of higher-order abstract syntax; like a number of other approaches they split the transition relation into two relations to handle scope extrusion. Their formalisations employ the meta-logic $\mathsf{FOL}^{\Delta\nabla}$ which forms the basis of the Abella theorem prover, and similar specifications have been used as the basis for verification of properties of the π-calculus in Abella [Baelde et al. 2014].

Accattoli [2012] adapts Huet's Coq formalisation of residuals from de Bruijn indices to Abella's higher-order abstract syntax and nominal quantifier ∇, yielding a significant simplification of Huet's proof. Accattoli also proves the cube lemma directly, rather than introducing an intermediate prism theorem.

Nominal Isabelle The Nominal Datatype Package extension to Isabelle/HOL supports the Gabbay-Pitts style "nominal" approach to abstract syntax modulo name-binding [Gabbay and Pitts 2002], and has been used in several formalisations. Two early contributions using similar ideas predate its development: Röckl [2001] formalised the syntax of π-calculus and α-equivalence in Isabelle/HOL. Gabbay [2003] described how to use Gabbay-Pitts nominal abstract syntax to represent the π-calculus, without giving a mechanised formalisation or proofs of properties.

Bengtson and Parrow [2009] report on an extensive formalisation in Nominal Isabelle, including inversion principles up to structural congruence, properties of strong and weak bisimulation, and a proof that an axiomatisation of strong late bisimilarity is sound and complete. They use a single inductively-defined transition relation, whose third argument is a sum type allowing either an ordinary process or a residual process with a distinguished bound name.

CLF Cervesato et al. [2002] formalise synchronous and asynchronous versions of π-calculus in the Concurrent Logical Framework (CLF), and Watkins et al. 2008 develop a static type system and operational semantics modeled on that of Gordon and Jeffrey 2003 for checking correspondence properties of protocols specified in the π-calculus. CLF employs higher-order abstract syntax, linearity and a monadic encapsulation of certain linear constructs that can identify objects such as traces up to causal equivalence. Thus, CLF's π-calculus encodings naturally induce equivalences on traces satisfying commuting conversions among synchonous operations. However, a non-trivial effort appears necessary to compare CLF's notion of trace equivalence with others, because traces are quotiented by a definitional equality by default and there is no explicit notion of concurrency or residuation.

Agda Orchard and Yoshida [2016] present a translation from a functional language with effects to a π-calculus with session types and verify some type-preservation properties of the translation in Agda.

6. Conclusions and future work

To the best of our knowledge, we are the first to report on a mechanised formalisation of concurrency, residuation and causal equivalence for the π-calculus. We employed de Bruijn indices to represent binders and names. Formalisations of λ-calculi often employ this technique, but to our knowledge only Orchard and Yoshida report a similar approach for a π-calculus formalisation. Whilst de Bruijn indices incur a certain level of administrative overhead, dependent types helps tame their complexity: many invariants are automatically checked by the type system rather than requiring additional explicit reasoning.

Our work appears to be the first to align the notion of "proved transitions" from Boudol and Castellani's work on CCS with "transition proofs" in the π-calculus. This hinges on the capability to manipulate and perform induction or recursion over derivations, and means we can leverage dependent typing so that residuation is defined only for concurrent transitions, rather than on all pairs of transitions. It is worth noting that while CLF's approach to encoding π-calculus automatically yields an equivalence on traces, it is unclear (at least to us) whether this equivalence is similar to the one we propose, or whether such traces can be manipulated explicitly as proof objects if desired.

The most notable aspect of our development is the generalised diamond lemma, which allows causally equivalent traces to have target states which are not equal "on the nose" but only up to a precise braiding which captures how binders were reordered. These braidings are more explicit in a de Bruijn indices setting, since free as well as bound names must be rewired when binders are transposed. Generalised cofinality may be relevant to modelling concurrency in other languages where concurrent transitions have effects which commute only up to some equivalence relation, such as dynamic memory allocation.

6.1. *Future work*

One possible future direction would be to explore trace structures explicitly quotiented by causal equivalence, such as dependence graphs [Mazurkiewicz 1987], event structures [Boudol and Castellani 1989], or rigid families [Cristescu et al. 2015]. We are also interested in extending our approach to accommodate structural congruences, and in understanding whether ideas from homotopy type theory [Univalent Foundations Program 2013], such as quotients or higher inductive types, could be applied to ease reasoning about π-calculus traces modulo causal equivalence and structural congruence.

Acknowledgements

We are grateful to our colleagues in the Programming Languages Interest Group at Edinburgh for useful discussions, to Vít Šefl for assistance with the Agda formalisation, and to the anonymous referees for comments on the paper. Effort sponsored by the Air Force Office of Scientific Research, Air Force Material Command, USAF, under grant number FA8655-13-1-3006, and EPSRC, grant number EP/K034413/1. The U.S. Government and University of Edinburgh are authorized to reproduce and distribute reprints for their purposes notwithstanding any copyright notation thereon.

References

Accattoli, B. (2012). Proof pearl: Abella formalization of λ-calculus cube property. In Hawblitzel, C. and Miller, D., editors, *Certified Programs and Proofs*, volume 7679 of *Lecture Notes in Computer Science*, pages 173–187. Springer Berlin Heidelberg.

Affeldt, R. and Kobayashi, N. (2008). A coq library for verification of concurrent programs. *Electron. Notes Theor. Comput. Sci.*, 199:17–32.

Aït Mohamed, O. (1995). Mechanizing a pi-calculus equivalence in hol. In *TPHOLs*, pages 1–16. Springer-Verlag.

Angiuli, C., Morehouse, E., Licata, D. R., and Harper, R. (2014). Homotopical patch theory. In *Proceedings of the 19th ACM SIGPLAN International Conference on Functional Programming*, ICFP '14, pages 243–256, New York, NY, USA. ACM.

Baelde, D., Chaudhuri, K., Gacek, A., Miller, D., Nadathur, G., Tiu, A., and Wang, Y. (2014). Abella: A system for reasoning about relational specifications. *Journal of Formalized Reasoning*, 7(2).

Bengtson, J. and Parrow, J. (2009). Formalising the pi-calculus using nominal logic. *Logical Methods in Computer Science*, 5(2:16).

Boreale, M. and Sangiorgi, D. (1998). A fully abstract semantics for causality in the π-calculus. *Acta Inf.*, 35(5):353–400.

Boudol, G. and Castellani, I. (1989). Permutation of transitions: An event structure semantics for CCS and SCCS. In Bakker, J., Roever, W.-P., and Rozenberg, G., editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, volume 354 of *LNCS*, pages 411–427. Springer.

Bucalo, A., Honsell, F., Miculan, M., Scagnetto, I., and Hofmann, M. (2006). Consistency of the theory of contexts. *J. Funct. Program.*, 16(3):327–372.

Cervesato, I., Pfenning, F., Walker, D., and Watkins, K. (2002). A concurrent logical framework ii: Examples and applications. Technical Report CMU-CS-02-102, Carnegie Mellon University.

Cristescu, I., Krivine, J., and Varacca, D. (2013). A compositional semantics for the reversible pi-calculus. In *LICS*, pages 388–397.

Cristescu, I. D., Krivine, J., and Varacca, D. (2015). *Theoretical Aspects of Computing - ICTAC 2015: 12th International Colloquium, Cali, Colombia, October 29-31, 2015, Proceedings*, chapter Rigid Families for CCS and the π-calculus, pages 223–240. Springer International Publishing.

Curry, H. B. and Feys, R. (1958). *Combinatory Logic*, volume 1 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, Amsterdam, Holland.

Danos, V. and Krivine, J. (2004). Reversible communicating systems. In Gardner, P. and Yoshida, N., editors, *CONCUR*, volume 3170 of *LNCS*, pages 292–307. Springer.

de Bruijn, N. (1972). Lambda-calculus notation with nameless dummies: a tool for automatic formula manipulation with application to the Church-Rosser theorem. *Indagationes Mathematicae*, 34(5):381–392.

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

Despeyroux, J. (2000). A higher-order specification of the pi-calculus. In *IFIP TCS*, number 1872 in LNCS, pages 425–439. Springer-Verlag.

Gabbay, M. J. (2003). The pi-calculus in FM. In Kamareddine, F., editor, *Thirty-five years of Automating Mathematics*, volume 28 of *Kluwer applied logic series*, pages 247–269. Kluwer.

Gabbay, M. J. and Pitts, A. M. (2002). A new approach to abstract syntax with variable binding. *Formal Aspects of Computing*, 13:341–363.

Gay, S. J. (2001). A framework for the formalisation of pi calculus type systems in isabelle/hol. In *TPHOLs*, pages 217–232. Springer-Verlag.

Gordon, A. D. and Jeffrey, A. (2003). Typing correspondence assertions for communication protocols. *Theor. Comput. Sci.*, 300(1-3):379–409.

Hirschkoff, D. (1997). A full formalisation of pi-calculus theory in the calculus of constructions. In *TPHOLs*, pages 153–169.

Hirschkoff, D. (1999). Handling substitutions explicitly in the pi-calculus. In *Proceedings of the Second International Workshop on Explicit Substitutions: Theory and Applications to Programs and Proofs*.

Honsell, F., Miculan, M., and Scagnetto, I. (2001). π-calculus in (co)inductive-type theory. *Theor. Comput. Sci.*, 253(2):239–285.

Huet, G. P. (1994). Residual theory in λ-calculus: A formal development. *Journal of Functional Programming*, 4(3):371–394.

Lanese, I., Mezzina, C. A., and Stefani, J.-B. (2010). Reversing higher-order pi. In *CONCUR*, pages 478–493. Springer-Verlag.

Lévy, J.-J. (1980). Optimal reductions in the lambda-calculus. In Seldin, J. P. and Hindley, J. R., editors, *To H. B. Curry: Essays in Combinatory Logic, Lambda Calculus and Formalism*, pages 159–191. Academic Press.

Mazurkiewicz, A. (1987). Trace theory. In *Advances in Petri Nets 1986, Part II on Petri Nets: Applications and Relationships to Other Models of Concurrency*, number 255 in LNCS, pages 279–324. Springer-Verlag.

Melham, T. F. (1994). A mechanized theory of the π-calculus in HOL. *Nordic J. of Computing*, 1(1):50–76.

Milner, R. (1999). *Communicating and mobile systems: the π calculus.* Cambridge University Press.

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

Norell, U. (2009). Dependently typed programming in Agda. In *Advanced Functional Programming*, volume 5832 of *LNCS*, pages 230–266. Springer.

Orchard, D. and Yoshida, N. (2016). Effects as sessions, sessions as effects. In *POPL 2016*. ACM.

Perera, R., Acar, U. A., Cheney, J., and Levy, P. B. (2012). Functional programs that explain their work. In *ICFP*, pages 365–376. ACM.

Perera, R. and Cheney, J. (2015). Proof-relevant pi-calculus. In Cervesato, I. and Chaudhuri, K., editors, Proceedings Tenth International Workshop on *Logical Frameworks and Meta Languages: Theory and Practice,* Berlin, Germany, 1 August 2015, volume 185 of *Electronic Proceedings in Theoretical Computer Science*, pages 46–70. Open Publishing Association.

Röckl, C. (2001). A first-order syntax for the pi-calculus in isabelle/hol using permutations. *Electr. Notes Theor. Comput. Sci.*, 58(1):1–17.

Röckl, C. and Hirschkoff, D. (2003). A fully adequate shallow embedding of the π-calculus in isabelle/hol with mechanized syntax analysis. *J. Funct. Program.*, 13(2):415–451.

Röckl, C., Hirschkoff, D., and Berghofer, S. (2001). Higher-order abstract syntax with induction in isabelle/hol: Formalizing the pi-calculus and mechanizing the theory of contexts. In *FOSSACS*, pages 364–378. Springer-Verlag.

Sangiorgi, D. and Walker, D. (2001). *The Pi-Calculus - a theory of mobile processes.* Cambridge University Press.

Stark, E. W. (1989). Concurrent transition systems. *Theoretical Computer Science*, 64(3):221–269.

Tiu, A. and Miller, D. (2010). Proof search specifications of bisimulation and modal logics for the π-calculus. *ACM Trans. Comput. Logic*, 11(2):13:1–13:35.

Univalent Foundations Program, T. (2013). *Homotopy Type Theory: Univalent Foundations of Mathematics.* `http://homotopytypetheory.org/book`, Institute for Advanced Study.

Watkins, K., Cervesato, I., Pfenning, F., and Walker, D. (2008). Specifying properties of concurrent computations in CLF. *Electron. Notes Theor. Comput. Sci.*, 199:67–87.

Appendix A. Agda module structure

Figure 18 summarises the module structure of the Agda formalisation.

Utilities	
`Ext`	Extensions to Agda library, `https://github.com/rolyp/agda-stdlib-ext`
Core modules	
`Action`	Actions a
`Action.Concur`	Concurrent actions $a \smile a'$; residuals a/a'
`Action.Seq`	Action sequences $\boldsymbol{a}$
`Braiding.Proc`	Bound braids $\phi : P \mathbin{\bar{\times}} P'$
`Braiding.Transition`	Residuals t/ϕ and ϕ/t
`Name`	Contexts Γ; names x
`Proc`	Processes P
`ProofRelevantPi`	Include everything; compile to build project
`ProofRelevantPiCommon`	Common imports from standard library
`Ren`	Renamings $\rho : \Gamma \longrightarrow \Gamma'$
`Ren.Properties`	Additional properties relating to renamings
`Transition`	Transitions $t : P \xrightarrow{a} R$
`Transition.Concur`	Concurrent transitions $t \smile t'$; residuals t/t'
`Transition.Concur.Cofinal`	Cofinality witnesses γ
`Transition.Concur.Cofinal.Transition`	Residuals t/γ and γ/t
`Transition.Seq`	Transition sequences
`Transition.Seq.Cofinal`	Residuals $\boldsymbol{t}/\gamma$ and $\gamma/\boldsymbol{t}$; permutation equivalence $\alpha : \boldsymbol{t} \simeq \boldsymbol{u}$
`Transition.Seq.Cofinal.Cofinal`	Proof that $\boldsymbol{t}/\gamma$ and $\gamma/\boldsymbol{t}$ are (heterogeneously) cofinal
Common sub-modules	
`.Ren`	Renaming lifted to entity defined in parent module

Fig. 18: Module overview, release `0.2`