Multiparty Symmetric Sum Types

Lasse Nielsen
DIKU, University of Copenhagen
lnielsen@diku.dk

Nobuko Yoshida
Imperial College London
yoshida@doc.ic.ac.uk

Kohei Honda
Queen Mary, University of London
kohei@dcs.qmul.ac.uk

This paper extends the multiparty asynchronous session types to symmetric sumtypes, which can type non-deterministic orchestration choice behaviours. While the original branching in the session types requires one participant to decide how to proceed by sending a label, with symmetric sumtypes the choice can be made in a non-deterministic way by synchronisation between the participants in a multiparty session. The motivation for synchronisation comes from natural and concise modelling of social interaction and cooperation in healthcare scenarios in the Process Matrix. The behaviour of synchronisation is represented by a new synchronise process constructor, which is typed by symmetric sumtypes. Finally we show that symmetric sumtypes can be erased into the original branching types with the help of conductor processes, preserving typability and operational semantics.

1 Introduction

Social interactions [1] are interactions among people as well as human organisations, as opposed to interactions among computer processes. Social interactions include workflow models and various cooperation models, and serve as a basis of many application-level protocols in computing such as healthcare and financial protocols. Just like their computer counterparts, social interactions are often highly structured, coming from the underlying social, economic and legal concerns. There is richness in social interaction structures, stemming from diverse collaborative patterns human and human organisations can exhibit. One of such patterns is for multiple participants in an interaction to collectively decide on one of the possible choices, as found in many implicit and explicit group-based agreement. Essence of this interaction pattern is, inside a conversation, there is a phase when a group of participants collectively wish to decide on one of the possible choices, and when they decide, they collectively commit to one option. One might call such an interaction pattern, *symmetric synchronisation*.

Motivated by examples from practice, this paper aims to distill the essence of such a symmetric synchronisation as an interaction primitive extending the π-calculus, and explores its properties using the type theory of the π-calculus. Our starting point is formal modelling, specification and verification of social interaction patterns in the healthcare domain, Clinical Practice Guideline (CPG) [11], a detailed description of medical treatment procedures, practised globally with variations. The CPG instance which motivated the present study is the specification called *Process Matrix* [12], where a workflow uses notions of actions on shared information storage, with different types of execution restrictions. In a close look, it includes a subtle form of collaborative actions, where users cooperate in order to complete all the actions. To explain its central structure, we give a simple example of a workflow from the Process Matrix.

This example consists of three participants: A doctor (D), a nurse (N) and a patient (P). The doctor and the nurse need to register and inspect the patient, therefore they need to perform three actions: Obtain the patient data (Data), schedule a time for the doctor to inspect the patient (Schedule) and for the doctor to actually inspect the patient (Inspect). The actions can be split in four different ways between the doctor and the nurse as illustrated in Figure 1. In Case 1, the doctor performs all the actions. In Case 2, the

Figure 1 Cases in the cooperation example

	Actions		
	Data	Schedule	Inspect
Case 1	D	D	D
Case 2	N	D	D
Case 3	D	N	D
Case 4	N	N	D

D: Doctor
N: Nurse
P: Patient

Data: Obtain patient data
Schedule: Schedule inspection
Inspect: Perform inspection

nurse obtains the patient data and the doctor schedules and performs the inspection. In Case 3 the doctor obtains the date, the nurse schedules the inspection and the doctor performs the inspection. In Case 4 the nurse obtains the data and schedules the inspection, so the doctor only has to perform the inspection. Notice that only the doctor is allowed to inspect the patient, but both the nurse and the doctor can get the data and schedule the inspection.

In this way, each participant need to perform a different combination of actions depending on which case is chosen: thus all participants need to *agree* on this choice, in order to make the cooperation work. If we encode this behaviour with asymmetric choice where one party selects which option is chosen (as found in branching/selection primitives [10, 4]) then it is necessary to let, for example, the doctor make the choice, who in turn tells the nurse and the patient to follow that choice using branch selection. Unfortunately this does not faithfully represent the target situation, since in practice, both the nurse and the patient will be able to influence the decision. Therefore, to model such a behaviour formally as a process, we introduce a *symmetric form of synchronisation* which allows us to represent situations where a decision is made collectively.

Our aim is to accurately capture practical collaboration scenarios such as CPG so that the resulting models serve as a basis of computer-based assistance of such a procedure with static validation. For this purpose we incorporate this primitive as part of *typed multiparty sessions*, in the sense of [2, 5]. In this framework, different groups of principals will participate in different threads of multiparty sessions, consisting of both standard asymmetric communications and symmetric synchronisations. These sessions are abstracted as types, enabling type-based validation leading to type and communication safety.

The use of types also enables us an organised analysis of this primitive. In particular we show this primitive can be embedded into an asymmetric form of branching/selection as in the original multiparty sessions [2, 5] preserving types and dynamics. While the idea of the encoding is intuitive, the whole translation procedure is non-trivial, requiring a generation of processes from a global type and delicate embeddings from symmetric sum types to branching types respecting global interaction structures, crucially exploiting the type structure. The representation makes use of a *conductor process* for each session, which conducts the synchronisations by receiving synchronisation input from each participant and sends the chosen case back. The resulting translation for the conductor introduces exponentially more branching cases (e.g. 8^3 more for the running example), demonstrating the usefulness of the symmetric sum types for compact and efficient multiparty symmetric synchronising session communications.

There are existing studies on self-synchronisations and broadcast synchronisations [3, 9], and encodability results of untyped asymmetric, directed sums [7, 6], but the symmetric sums we study are different because they are type driven and allow all the participants to influence the choice equally, and we know of no study on session-based formalisations of a symmetric choice primitive, its type discipline, or type-directed encodings of it. The non-encodability of the *mixed-choice* π-calculus in the *separate-*

Figure 2 The process language

$P ::= \overline{a}[2..\mathtt{n}](\tilde{s}).P$	session request	$e ::= v$	values
$\mid a[\mathtt{p}](\tilde{s}).P$	session acceptance	$\mid \mathsf{rand}\{v_i\}_{i\in I}$	random selection
$\mid s!\langle\tilde{e}\rangle;P$	value sending	$\mid e \mathsf{\ and\ } e'$	boolean conjunction
$\mid s?(\tilde{x});P$	value reception	$\mid \mathsf{not}\ e$	boolean negation
$\mid s!\langle\langle\tilde{s}\rangle\rangle;P$	session delegation	$\mid \ldots$	other expressions
$\mid s?((\tilde{s}));P$	session reception		
$\mid s \lhd l;P$	label selection	$D ::= \{X_i(\tilde{x}_i\tilde{s}_i) = P_i\}_{i\in I}$	declaration
$\mid s \rhd \{l : P_l\}_{l\in L}$	label branching		
$\mid \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q$	conditional	$v ::= a$	channels
$\mid P \mid Q$	parallel composition	$\mid \mathsf{true}$	boolean true
$\mid 0$	inaction	$\mid \mathsf{false}$	boolean false
$\mid (\nu n)P$	hiding		
$\mid \mathsf{def}\ D\ \mathsf{in}\ P$	recursion	$h ::= l$	label in-transit
$\mid X\langle\tilde{e}\tilde{s}\rangle$	process call	$\mid \tilde{v}$	value in-transit
$\mid s : \tilde{h}$	message queue	$\mid \tilde{s}$	session in-transit
$\mid \mathtt{sync}_{\tilde{s},n}\{l : P_l\}_{l\in L}$	synchronisation		

choice (asymmetric, uni-directed sum) π-calculus in [8] may not directly relate to our translation result since the target sum is of a different kind and the result in [8] is proved in the *untyped* calculi (though our translation does use an extra agent, thus not fulfilling the restriction set out in [8]).

This paper reports the formalisation of the syntax, semantics and typing of the described synchronisation and define a type-directed and semantics-preserving encoding of the symmetric synchronisation primitive in the π-calculus with multiparty sessions. The main results are Subject reduction (Theorem 3.2) which attests the consistency of our type theory, and Type and Semantic preservation of the encoding (Theorem 4.4 and Corollary 4.6) demonstrating the feasible implementability and significance of the new primitive. The example outlined above is used throughout the subsequent sections to illustrate the syntax, semantics, typing system and synchronisation erasure mapping of the new primitive.

In the rest of the paper, Section 2 introduces the syntax and semantics of the new synchronisation primitive. Section 3 defines the types and the typing system. Section 4 studies the synchronisation encoding. Section 5 concludes with future work. Appendix lists the omitted definitions and proofs.

2 Processes with Synchronisation

This section introduces a new communication primitive, `sync`, which represents the symmetric form of synchronisation reflecting the specific form of social interaction where a common decision is made among two or more parties. We extend the calculus with asynchronous multiparty sessions developed in [5]. Due to the space limitation, we omit explanations for the part of syntax, semantics and types that are identical with [5].

Syntax The entire process language is defined in Figure 2. The new constructor $\mathtt{sync}_{\tilde{s},n}\{l : P_l\}_{l\in L}$ is interpreted as the process participating in a plenum decision between all the n processes in the session $\tilde{s}$ reaching a common decision h for some $h \in L$. Afterwards the process proceeds as described in P_h. This means that all the processes in each session must have a `sync` constructor and they will each use

Figure 3 Small step semantics of process language

$$\text{Link}\ \frac{}{\vdash \overline{a}[2..n](\tilde{s}).P_1 | a[2](\tilde{s}).P_2 | \ldots | a[n](\tilde{s}).P_n \to (\nu \tilde{s})(P_1|P_2|\ldots|P_n|s_1:\emptyset|\ldots|s_m:\emptyset)}$$

$$\text{Send}\ \frac{\vdash \tilde{e} \downarrow \tilde{v}}{\vdash s!\langle \tilde{e}\rangle; P | s : \tilde{h} \to P | s : \tilde{h}\cdot\tilde{v}} \qquad \text{Deleg}\ \frac{}{\vdash s!\langle\langle \tilde{t}\rangle\rangle; P | s : \tilde{h} \to P | s : \tilde{h}\cdot\tilde{t}} \qquad \text{Label}\ \frac{}{\vdash s \triangleleft l; P | s : \tilde{h} \to P | s : \tilde{h}\cdot l}$$

$$\text{Recv}\ \frac{}{\vdash s?(\tilde{x}); P | s : \tilde{\tilde{v}}\cdot h \to P[\tilde{v}/\tilde{x}] | s : \tilde{h}} \qquad \text{SRec}\ \frac{}{\vdash s?((\tilde{t})); P | s : \tilde{t}\cdot h \to P | s : \tilde{h}}$$

$$\text{Brach}\ \frac{j \in I}{\vdash s \triangleright \{l_i : P_i\}_{i\in I} | s : l_j \cdot h \to P_j | s : \tilde{h}} \qquad \text{IfT}\ \frac{\vdash e \downarrow \mathsf{true}}{\vdash \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \to P}$$

$$\text{IfF}\ \frac{\vdash e \downarrow \mathsf{false}}{\vdash \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \to Q} \qquad \text{Def}\ \frac{\vdash \tilde{e} \downarrow \tilde{v} \quad \vdash X\langle \tilde{x}\tilde{s}\rangle = P \in D}{\vdash \mathsf{def}\ D\ \mathsf{in}\ X\langle \tilde{e}\tilde{s}\rangle | Q \to \mathsf{def}\ D\ \mathsf{in}\ P[\tilde{v}/\tilde{x}] | Q}$$

$$\text{Scop}\ \frac{\vdash P \to P'}{\vdash (\nu n)P \to (\nu n)P'} \qquad \text{Par}\ \frac{\vdash P \to P'}{\vdash P|Q \to P'|Q} \qquad \text{Defin}\ \frac{\vdash P \to P'}{\vdash \mathsf{def}\ D\ \mathsf{in}\ P \to \mathsf{def}\ D\ \mathsf{in}\ P'}$$

$$\text{Str}\ \frac{\vdash P \equiv P' \quad \vdash P \to Q \quad \vdash Q \equiv Q'}{\vdash P' \to Q'} \qquad \text{Sync}\ \frac{h \in \bigcap_{i=1}^{n} L_i}{\mathtt{sync}_{\tilde{s},n}\{l : P_{1l}\}_{l\in L_1} \mid \ldots \mid \mathtt{sync}_{\tilde{s},n}\{l : P_{nl}\}_{l\in L_n} \to P_{1h} \mid \ldots \mid P_{nh}}$$

the same branch, even though the chosen branch may vary between executions or even between `syncs` in each execution.

It can be discussed if each process must use the same set of labels in the `sync`, but this is not enforced as it results in a richer process language. This is also mentioned in the example at the end of this section. Of course the processes cannot perform the synchronisation if they do not all share a common label, in which case the processes will be stuck. Therefore the type system will ensure that this does not occur.

The semantic properties that are required for `sync` is that all the processes must participate in the `sync`, and if they step all must use the same label. This is obtained by letting all the processes step at the same time in a synchronous way, ensuring that all the properties can be checked during the step and therefore avoiding extra environment clutter. This approach results in the stepping rule:

$$\text{Sync}\ \frac{h \in \bigcap_{i=1}^{n} L_i}{\mathtt{sync}_{\tilde{s},n}\{l : P_{1l}\}_{l\in L_1} \mid \ldots \mid \mathtt{sync}_{\tilde{s},n}\{l : P_{nl}\}_{l\in L_n} \to P_{1h} \mid \ldots \mid P_{nh}}$$

Notice that we need to know how many participants are in the session in order to know when the synchronisation can step. This is done by including the number of processes in the syntax of `sync`, so it can be used in the stepping rule, and verified by the type system. Adding the above stepping rule to the existing calculi results in the small-step semantics in Figure 3. For explanations of the rest of the syntax and operational semantics, see [5].

Healthcare Cooperation Example We motivate the new communication primitive through a simple example. Recall the cooperation example from Introduction. There were four ways to divide the actions to be performed between the doctor and the nurse, as illustrated in Figure 1.

We first explain the problem when representing the same situation without using `sync`. As we can see from the example, there is no rigorous way to decide which of the four cases will occur, as well as who will be the principal decision maker: we could let the doctor decide between the cases, and then we get the processes in Figure 4, if we only use the language from [5]. We could also let the nurse or even the patient decide between the cases, but none of these implementations captures the nature of

Figure 4 Healthcare Example without `sync`

$P_D = a[2](p1, p2, d, n).$
if ... then $p1 \triangleleft \text{case1}$;
$d?(\text{data}); p1!\langle e_{\text{schedule}}\rangle;$
$p1!\langle e_{\text{result}}\rangle; 0$
else if ... then $p1 \triangleleft \text{case2}$;
$p1!\langle e_{\text{schedule}}\rangle; p1!\langle e_{\text{result}}\rangle; 0$
else if ... then $p1 \triangleleft \text{case3}$;
$d?(\text{data}).p1!\langle e_{\text{result}}\rangle; 0$
else $p1 \triangleleft \text{case4}$;
$p1!\langle e_{\text{result}}\rangle; 0$

$P_N = a[3](p1, p2, d, n).n \triangleright \{$
case1 : 0,
case2 : $n?(\text{data}); 0$,
case3 : $p2!\langle e_{\text{schedule}}\rangle; 0$,
case4 : $n?(\text{data});$
$p2!\langle e_{\text{schedule}}\rangle; 0$
$\}$

$P_P = \overline{a}[2,3](p1, p2, d, n).p1 \triangleright \{$
case1 : $n \triangleleft \text{case1}; d!\langle e_{\text{data}}\rangle;$
$p1?(\text{schedule}); p1?(\text{result}); 0$,
case2 : $n \triangleleft \text{case2}; n!\langle e_{\text{data}}\rangle;$
$p1?(\text{schedule}); p1?(\text{result}); 0$,
case3 : $n \triangleleft \text{case3}; d!\langle e_{\text{data}}\rangle;$
$p2?(\text{schedule}); p1?(\text{result}); 0$,
case4 : $n \triangleleft \text{case4}; n!\langle e_{\text{data}}\rangle;$
$p2?(\text{schedule}); p1?(\text{result}); 0\}$

the cooperation between the doctor and the nurse where they reach a common decision, because it is impossible to know who takes the *initiative*. It will also be a problem to write the unspecified parts of P_D as it is meant to describe a human decision, thus it is necessary to use non-deterministic expressions as random.

We now express the same example using the new synchronisation constructor `sync`. This constructor can be used to select one of the four cases for all the processes, resulting in the processes in Figure 5. This means that we have not violated the intent of the cooperation, but it also means that the decision has not been specified. The choice can however be further specified, by letting the participants narrow the choices depending on some conditions. For example we could rewrite the nurse process to the process in Figure 6 to represent more fine-grained behaviours with conditionals. It is, however, necessary to ensure that each `sync` has at least one shared label for all the participants in order to avoid a stuck state.

3 Symmetric Sum Types

This section introduces a new type constructor for symmetric synchronisation, which we call *symmetric sum types*, enabling type-checking of processes using the `sync` constructor.

Types We start by defining the global types G in Figure 7, which are used to describe session protocols among session participants. each communication constructor apart from symmetric synchronisation

Figure 5 Healthcare Example using `sync`

$P_D = a[2](p1, p2, d, n).$
$\text{sync}_{(p1,p2,d,n),3}\{$
case1 : $d?(\text{data}); p1!\langle e_{\text{schedule}}\rangle;$
$p1!\langle e_{\text{result}}\rangle; 0$,
case2 : $p1!\langle e_{\text{schedule}}\rangle;$
$p1!\langle e_{\text{result}}\rangle; 0$,
case3 : $d?(\text{data}); p1!\langle e_{\text{result}}\rangle; 0$,
case4 : $p1!\langle e_{\text{result}}\rangle; 0\}$

$P_N = a[3](p1, p2, d, n).$
$\text{sync}_{(p1,p2,d,n),3}\{$
case1 : 0,
case2 : $n?(\text{data}); 0$,
case3 : $p2!\langle e_{\text{schedule}}\rangle; 0$,
case4 : $n?(\text{data});$
$p2!\langle e_{\text{schedule}}\rangle; 0\}$

$P_P = \overline{a}[2,3](p1, p2, d, n).\text{sync}_{(p1,p2,d,n),3}\{$
case1 : $d!\langle e_{\text{data}}\rangle; p1?(\text{schedule});$
$p1?(\text{result}); 0$,
case2 : $n!\langle e_{\text{data}}\rangle; p1?(\text{schedule});$
$p1?(\text{result}); 0$,
case3 : $d!\langle e_{\text{data}}\rangle; p2?(\text{schedule});$
$p1?(\text{result}); 0$,
case4 : $n!\langle e_{\text{data}}\rangle; p2?(\text{schedule});$
$p1?(\text{result}); 0\}$

Figure 6 Example of Partially Specialised Nurse Process

$P_N = a[3](p1, p2, d, n)$.if busy
then $\texttt{sync}_{(p1,p2,d,n),3}\{$ case1 : 0, case2 : $n?(\text{data}); 0$, case3 : $p2!\langle e_{\text{schedule}}\rangle; 0\}$
else $\texttt{sync}_{p1,p2,d,n}\{$ case2 : $n?(\text{data}); 0$, case3 : $p2!\langle e_{\text{schedule}}\rangle; 0$, case4 : $n?(\text{data}); p2!\langle e_{\text{schedule}}\rangle; 0\}$

describes a single communication action with the given type from one sender to one receiver. This is the same as in the original system [5] except for two changes: We have removed the constructor for parallel protocols, as it does not affect the expressiveness of the types, and the symmetric sum type constructor $\{l : G_l\}_{l \in L; L'}$ is added. The symmetric sum type is similar to the branch type & and its dual, selection type $\oplus$, but there is no sender, receiver or channel given in the constructor. The sum type represents a synchronisation where the labels are taken from L and L'. The labels in L are optional, but the labels in L' are mandatory and will be underlined when concrete types are written. For example the type $\{l : T_l\}_{l \in \{l1\};\{l2\}} = \{l1 : T_{l1}, \underline{l2} : T_{l2}\}$. To ensure that the synchronisation can step we require that $L' \neq \emptyset$. The global types use the message types U and simple types S which are also defined in Figure 7. The only difference from the definition of U and S in [5] is that U now includes information about the number of session channels m and the number of session processes n in a local type T.

The local types T are defined in Figure 7. They describe the communication performed by a single process. Therefore the "from process to process on channel" syntax is simply changed to sending or receiving on a channel. The difference from [5] is that the symmetric sum type constructor $\{l : T_l\}_{l \in L; L'}$ is added. The symmetric sum type is similar to the branch type, but no channel is given in the constructor. The sum type represents a synchronisation between all the processes in the session, where the labels are taken from L and L'. Like in the global types, the labels in L are optional, but the labels in L' are mandatory and will be underlined when concrete types are written just like global types. Like global types, local types uses S and U from Figure 7 for the domain of simple and message types.

The projection from global types to local types can be found in Figure 8. The differences from the definition in [5] is that we have added a case for the symmetric sum type. Another key extension is the projection of the branching types where more cases are defined using the subtyping from [5, § 5] defined in Figure 16 in the appendix. In the original version of the projection, each branch had to be the same for all processes except the sender and receiver. In this version we only require that the common subtypes of all the projected branches have a maximum element with respect to the branching subtyping relation $\leq_{\text{sub}}$ from [5, § 5] defined in Figure 16 in the appendix. This is fulfilled whenever all the branches have a common subtype, but the projection is only defined when there is the maximum element.

Figure 7 The Domains used for Global and Local types

(Global Types)

$$
\begin{aligned}
G ::= {} & \mathtt{p} \to \mathtt{p}' : k\langle U\rangle.G' \\
\mid {} & \mathtt{p} \to \mathtt{p}' : k\{l_i : G_i\}_{i \in I} \\
\mid {} & \mu t.G \\
\mid {} & t \\
\mid {} & \mathsf{end} \\
\mid {} & \{l : G_l\}_{l \in L; L'} \quad (L' \neq \emptyset)
\end{aligned}
$$

(Message Types)

$$
\begin{aligned}
U ::= {} & \tilde{S} \\
\mid {} & T@(\mathtt{p}, m, n)
\end{aligned}
$$

(Simple Types)

$$
\begin{aligned}
S ::= {} & \mathsf{bool} \\
\mid {} & \mathsf{int} \\
\mid {} & \ldots \\
\mid {} & \langle G\rangle
\end{aligned}
$$

(Local Types)

$$
\begin{aligned}
T ::= {} & k!\langle U\rangle; T \\
\mid {} & k?\langle U\rangle; T \\
\mid {} & k \oplus \{l : T_l\}_{l \in L} \\
\mid {} & k \,\&\, \{l : T_l\}_{l \in L} \\
\mid {} & \mu t.T \\
\mid {} & t \\
\mid {} & \mathsf{end} \\
\mid {} & \{l : T_l\}_{l \in L; L'}
\end{aligned}
$$

Figure 8 Projection from Global to Local Types

$$
\begin{array}{rcl}
(\mathtt{p}_0 \to \mathtt{p}_1 : k\langle U\rangle.G')\upharpoonright \mathtt{p} & = & \begin{cases} m!\langle U\rangle;(G'\upharpoonright \mathtt{p}) & \text{if } \mathtt{p} = \mathtt{p}_0 \text{ and } \mathtt{p} \neq \mathtt{p}_0 \\ m?\langle U\rangle;(G'\upharpoonright \mathtt{p}) & \text{if } \mathtt{p} = \mathtt{p}_1 \\ G'\upharpoonright \mathtt{p} & \text{if } \mathtt{p} \neq \mathtt{p}_0 \text{ and } \mathtt{p} \neq \mathtt{p}_1 \end{cases} \\
(\mathtt{p}_1 \to \mathtt{p}_2 : k\{l_j : G_j\}_{j\in J})\upharpoonright \mathtt{p} & = & \begin{cases} k \oplus \{l_j : (G_j\upharpoonright \mathtt{p})\}_{j\in J} & \text{if } \mathtt{p} = \mathtt{p}_1 \neq \mathtt{p}_2 \\ k\&\{l_j : (G_j\upharpoonright \mathtt{p})\}_{j\in J} & \text{if } \mathtt{p} = \mathtt{p}_2 \neq \mathtt{p}_1 \\ \max_{\leq_{\mathtt{sub}}}\{T' \mid \forall j \in J.T' \leq_{\mathtt{sub}} (G_j\upharpoonright \mathtt{p})\} & \text{if } \mathtt{p} \neq \mathtt{p}_1 \text{ and } \mathtt{p} \neq \mathtt{p}_2 \end{cases} \\
(\{l : G_l\}_{l\in L;L'})\upharpoonright \mathtt{p} & = & \{l : (G_l\upharpoonright \mathtt{p})\}_{l\in L;L'} \\
(\mu t.G)\upharpoonright \mathtt{p} & = & \mu t.(G\upharpoonright \mathtt{p}) \quad t\upharpoonright \mathtt{p} = t \quad \mathsf{end}\upharpoonright \mathtt{p} = \mathsf{end}
\end{array}
$$

Note that since all the users of a session must participate in each synchronisation, there are no extra linearity or coherence constraints. The definitions just need to be updated. Thus all the branches of a symmetric sum must be linear and coherent.

We then define the global environment Γ containing the global types for shared channels and process variables, and the local type environment Δ containing the remaining session communication.

$$\Gamma ::= \emptyset \mid \Gamma, u : \langle G\rangle \mid \Gamma, X : \tilde{S}\tilde{T} \qquad \Delta ::= \emptyset \mid \Delta, \tilde{s} : T@(\mathtt{p}, n)$$

The Typing Judgement We are now ready to introduce the typing judgement which extends the typing from the original article[5] with symmetric sum types.

The typing rules are given in Figure 17 in the appendix. The rules are the same as the original type system from [5], except the rule for synchronisation and the local types now carry information about the number of participants n and channels. The number of participants n for synchronisation and local types are determined when we type the session initialisation.

$$\text{Mcast}\frac{\Gamma \vdash a : \langle G\rangle \quad \Gamma \vdash P \rhd \Delta, \tilde{s} : (G\upharpoonright 1)@(1, n) \quad |\tilde{s}| = \mathsf{max}(\mathsf{sid}(G)) \quad n = \mathsf{max}(\mathsf{pid}(G))}{\Gamma \vdash \overline{a}[2..\mathtt{n}](\tilde{s}).P \rhd \Delta}$$

where $\mathsf{sid}(G)$ denotes channels that appear in G and $\mathsf{pid}(G)$ denotes the participants that appear in G. Then the main Sync rule is defined as follows:

$$\text{Sync}\frac{\forall l \in L'' : \Gamma \vdash P_l \rhd \Delta, \tilde{s} : T_l@(\mathtt{p}, n) \quad L'' \subseteq L \cup L' \quad L' \subseteq L''}{\Gamma \vdash \mathtt{sync}_{\tilde{s},n}\{l : P_l\}_{l\in L''} \rhd \Delta, \tilde{s} : \{l : T_l\}_{l\in L;L'}@(\mathtt{p}, n)}$$

This rule allows all the labels in L and L', but only requires the mandatory labels L'.

Healthcare Cooperation Example We now go back to the example from Section 2, and explain how the types describe and verify social interaction and cooperation in a healthcare scenario. Recall the processes from Figure 5. To type $P_D \mid P_N \mid P_P$, we need a matching type-environment first. The processes uses the public channel a to create a session, so the environment must be of the form $\Gamma = a : \langle G\rangle$ for some global type G.

We will start by finding the type describing the interactions in Case 2. First the patient sends the data to the nurse, then the doctor schedules the inspection, so he sends the appointment to the patient. Finally

Figure 9 The Global Type G and its Projections for Healthcare Cooperation Example

$$
\begin{array}{llll}
G = \{ & G\restriction 1 = \{ & G\restriction 2 = \{ & G\restriction 3 = \{ \\
\quad \mathit{case}1 : \mathtt{1} \to \mathtt{2} : 3\langle \tilde{S}_{\text{data}} \rangle; & \quad \mathit{case}1 : 3!\langle \tilde{S}_{\text{data}} \rangle; & \quad \mathit{case}1 : 3?\langle \tilde{S}_{\text{data}} \rangle; & \quad \mathit{case}1 : \mathsf{end} \\
\qquad \mathtt{2} \to \mathtt{1} : 1\langle \tilde{S}_{\text{schedule}} \rangle; & \qquad 1?\langle \tilde{S}_{\text{schedule}} \rangle; & \qquad 1!\langle \tilde{S}_{\text{schedule}} \rangle; & \quad \underline{\mathit{case}2} : 4?\langle \tilde{S}_{\text{data}} \rangle; \\
\qquad \mathtt{2} \to \mathtt{1} : 1\langle \tilde{S}_{\text{result}} \rangle; & \qquad 1?\langle \tilde{S}_{\text{result}} \rangle; & \qquad 1!\langle \tilde{S}_{\text{result}} \rangle; & \qquad \mathsf{end} \\
\qquad \mathsf{end} & \qquad \mathsf{end} & \qquad \mathsf{end} & \quad \mathit{case}3 : 2!\langle \tilde{S}_{\text{schedule}} \rangle; \\
\quad \underline{\mathit{case}2} : \mathtt{1} \to \mathtt{3} : 4\langle \tilde{S}_{\text{data}} \rangle; & \quad \underline{\mathit{case}2} : 4!\langle \tilde{S}_{\text{data}} \rangle; & \quad \underline{\mathit{case}2} : 1!\langle \tilde{S}_{\text{schedule}} \rangle; & \qquad \mathsf{end} \\
\qquad \mathtt{2} \to \mathtt{1} : 1\langle \tilde{S}_{\text{schedule}} \rangle; & \qquad 1?\langle \tilde{S}_{\text{schedule}} \rangle; & \qquad 1!\langle \tilde{S}_{\text{result}} \rangle; & \quad \mathit{case}4 : 4?\langle \tilde{S}_{\text{data}} \rangle; \\
\qquad \mathtt{2} \to \mathtt{1} : 1\langle \tilde{S}_{\text{result}} \rangle; & \qquad 1?\langle \tilde{S}_{\text{result}} \rangle; & \qquad \mathsf{end} & \qquad 2!\langle \tilde{S}_{\text{schedule}} \rangle; \\
\qquad \mathsf{end} & \qquad \mathsf{end} & \quad \mathit{case}3 : 3?\langle \tilde{S}_{\text{data}} \rangle; & \qquad \mathsf{end} \} \\
\quad \mathit{case}3 : \mathtt{1} \to \mathtt{2} : 3\langle \tilde{S}_{\text{data}} \rangle; & \quad \mathit{case}3 : 3!\langle \tilde{S}_{\text{data}} \rangle; & \qquad 1!\langle \tilde{S}_{\text{result}} \rangle; & \\
\qquad \mathtt{3} \to \mathtt{1} : 2\langle \tilde{S}_{\text{schedule}} \rangle; & \qquad 2?\langle \tilde{S}_{\text{schedule}} \rangle; & \qquad \mathsf{end} & \\
\qquad \mathtt{2} \to \mathtt{1} : 1\langle \tilde{S}_{\text{result}} \rangle; & \qquad 1?\langle \tilde{S}_{\text{result}} \rangle; & \quad \mathit{case}4 : 1!\langle \tilde{S}_{\text{result}} \rangle; & \\
\qquad \mathsf{end} & \qquad \mathsf{end} & \qquad \mathsf{end} \} & \\
\quad \mathit{case}4 : \mathtt{1} \to \mathtt{3} : 4\langle \tilde{S}_{\text{data}} \rangle; & \quad \mathit{case}4 : 4!\langle \tilde{S}_{\text{data}} \rangle; & & \\
\qquad \mathtt{3} \to \mathtt{1} : 2\langle \tilde{S}_{\text{schedule}} \rangle; & \qquad 2?\langle \tilde{S}_{\text{schedule}} \rangle; & & \\
\qquad \mathtt{2} \to \mathtt{1} : 1\langle \tilde{S}_{\text{result}} \rangle; & \qquad 1?\langle \tilde{S}_{\text{result}} \rangle; & & \\
\qquad \mathsf{end} \} & \qquad \mathsf{end} \} & &
\end{array}
$$

the doctor inspects the patient, thus he sends the result to the patient. When the patient has id 1, the doctor has id 2 and the nurse has id 3 the described communication for Case 2 is described by the type

$$\mathtt{1} \to \mathtt{3} : 3\langle \tilde{S}_{\text{data}} \rangle; \mathtt{2} \to \mathtt{1} : 1\langle \tilde{S}_{\text{schedule}} \rangle; \mathtt{2} \to \mathtt{1} : 1\langle \tilde{S}_{\text{result}} \rangle; \mathsf{end}$$

Performing the same reasoning for Case 1, Case 3 and Case 4 and combining the result in a symmetric sum type results in the global type G in Figure 9. We underline the mandatory label, case2, in the global and local types. Notice that the global type clearly represents the protocol described in the matrix in Figure 1. We can then find the projections of G to find the local type for each process which is also given in Figure 9.

Proposition 3.1 (Example is well-typed)**.** $a : \langle G \rangle \vdash P_D \mid P_N \mid P_P \rhd \emptyset$.

PROOF: To show this we will need to type each case of each process with the matching case of the local type.

First the derivation for P_{Case1} and P_{Case3} (denoted by $\mathscr{D}(P_{\text{Case1}})$ and $\mathscr{D}(P_{\text{Case3}})$, respectively) are given as:

$$\text{Send}\dfrac{\Gamma \vdash e_{\text{data}} : \tilde{S}_{\text{data}} \qquad \text{Rcv}\dfrac{\text{Rcv}\dfrac{\text{Inact}\dfrac{}{\Gamma, \text{schedule} : \tilde{S}_{\text{schedule}}, \text{result} : \tilde{S}_{\text{result}} \vdash 0 \rhd (p,d,n) : \mathsf{end}@(1,3)}}{\Gamma, \text{schedule} : \tilde{S}_{\text{schedule}} \vdash p?(\text{result}); 0 \rhd (p,d,n) : 1?\langle \tilde{S}_{\text{result}} \rangle; \mathsf{end}@(1,3)}}{\Gamma \vdash p?(\text{schedule}); p?(\text{result}); 0 \rhd (p,d,n) : 1?\langle \tilde{S}_{\text{schedule}} \rangle; 1?\langle \tilde{S}_{\text{result}} \rangle; \mathsf{end}@(1,3)}}{\Gamma \vdash d!\langle e_{\text{data}} \rangle; p?(\text{schedule}); p?(\text{result}); 0 \rhd (p,d,n) : 2!\langle \tilde{S}_{\text{data}} \rangle; 1?\langle \tilde{S}_{\text{schedule}} \rangle; 1?\langle \tilde{S}_{\text{result}} \rangle; \mathsf{end}@(1,3)}$$

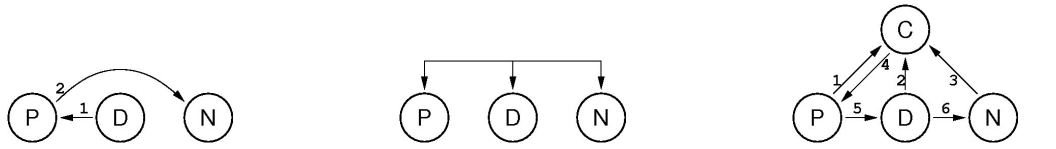
and then $\mathscr{D}(P_{\text{Case2}})$ and $\mathscr{D}(P_{\text{Case4}})$ are given as

$$\text{Send}\dfrac{\Gamma \vdash e_{\text{data}} : \tilde{S}_{\text{data}} \qquad \text{Rcv}\dfrac{\text{Rcv}\dfrac{\text{Inact}\dfrac{}{\Gamma, \text{schedule} : \tilde{S}_{\text{schedule}}, \text{result} : \tilde{S}_{\text{result}} \vdash 0 \rhd (p,d,n) : \mathsf{end}@(1,3)}}{\Gamma, \text{schedule} : \tilde{S}_{\text{schedule}} \vdash p?(\text{result}); 0 \rhd (p,d,n) : 1?\langle \tilde{S}_{\text{result}} \rangle; \mathsf{end}@(1,3)}}{\Gamma \vdash p?(\text{schedule}); p?(\text{result}); 0 \rhd (p,d,n) : 1?\langle \tilde{S}_{\text{schedule}} \rangle; 1?\langle \tilde{S}_{\text{result}} \rangle; \mathsf{end}@(1,3)}}{\Gamma \vdash n!\langle e_{\text{data}} \rangle; p?(\text{schedule}); p?(\text{result}); 0 \rhd (p,d,n) : 3!\langle \tilde{S}_{\text{data}} \rangle; 1?\langle \tilde{S}_{\text{schedule}} \rangle; 1?\langle \tilde{S}_{\text{result}} \rangle; \mathsf{end}@(1,3)}$$

Once we have proved the sub-result for each case, we can collect them using the Sync inference rule for P' to derive $\mathscr{D}(P')$ as follows:

$$\text{Sync}\dfrac{\mathscr{D}_{P_{\text{Case1}}} \quad \mathscr{D}_{P_{\text{Case2}}} \quad \mathscr{D}_{P_{\text{Case3}}} \quad \mathscr{D}_{P_{\text{Case4}}}}{\Gamma \vdash \mathtt{sync}_{p,d,n}\{\text{case1} : d!\text{data}.p?\text{schedule}.p?\text{result}.0, \ldots \text{ case4} : n!\text{data}.p?\text{schedule}.p?\text{result}.0\} \rhd (p,d,n) : (G\restriction 1)@(1,3)}$$

Figure 10 Communication structures



(a) Choice without `sync` (b) Choice using `sync` (c) Choice after `sync`-erasure

Then we can type P_P using the Mcast rule as $a : \langle G \rangle \vdash P_P \triangleright \emptyset$. The derivations for P_D and P_N are found similarly except that the Macc rule is used in stead of Mcast. Finally the derivations for each process are collected using the Conc rule twice which gives us that $a : \langle G \rangle \vdash P_N \mid P_D \mid P_P \triangleright \emptyset$. □

Subject Reduction We will now prove Subject reduction. To do that we will use the runtime typing syntax from [5] and we need to add a rule to the *type reduction* from [5] matching the `sync` case

$$\tilde{s} : \{\{l : T_{\mathtt{p}}, \ldots\}@(\mathtt{p}, n)\}_{\mathtt{p} \in \{1..n\}} \rightarrow \tilde{s} : \{T_{\mathtt{p}}@(\mathtt{p}, n)\}_{\mathtt{p} \in \{1..n\}}.$$

Theorem 3.2 (Subject Reduction)**.** $\Gamma \vdash P \triangleright_{\tilde{s}} \Delta$ and Δ coherent and $P \rightarrow P'$ implies $\Gamma \vdash P' \triangleright_{\tilde{s}} \Delta'$ where $\Delta = \Delta'$ or $\Delta \rightarrow \Delta'$.

PROOF: Induction on the derivation of $P \rightarrow P'$. See Appendix A.1.

4 From Symmetric Sumtypes to Conducted Branching

This section studies the erasure of symmetric synchronisation, which we hereafter simply call *erasure*. The erasure removes all occurrences of the `sync` constructor while preserving static and dynamic semantics, i.e. typability and reduction. The erasure works by inserting a conductor process and replaces all synchronisations with branching selected by the conductor process.

We will explain the basic idea using Figure 10, which shows the communication structure of three sets of processes involving a symmetric synchronisation. Figure 10(a) shows the communication between the processes without using `sync` in Figure 4, where the doctor chooses the case, sends the result to the patient who forwards it to the nurse. Figure 10(b) shows the communication between the processes using `sync` in Figure 5, where no messages are sent, but the synchronisation ensures that both the patient, doctor and nurse use the same case. Figure 10(c) shows the communication in the processes where the synchronisation has been removed in Figure 14, where first the patient, the doctor and the nurse sends the cases they can accept to the conductor, who chooses a common case and sends the selected case to the patient who forwards it to the doctor who forwards it to the nurse. Based on this idea, we translate the synchronisation and symmetric sumtypes into the original system [5], step by step as follows.

Step 1: Process Erasure Only well-typed processes are eligible for erasure, because conductor processes are generated from global types. Therefore the erasure $\mathscr{E}$ is defined by induction on the type derivation and the result is the final process. The erasure $\mathscr{E}$ is defined in Figure 11. There are three cases of special interest in the definition of $\mathscr{E}$.

Figure 11 Erasure of Synchronisation from Typing-Derivation

$$\mathscr{E}\left(\text{Mcast}\dfrac{\Gamma\vdash a:\langle G\rangle \quad \dfrac{\mathscr{D}_1}{\Gamma\vdash P\triangleright\Delta,\tilde{s}:(G[\mathtt{1}])@(\mathtt{1},n)} \quad \begin{array}{l}|\tilde{s}|=\mathsf{max}(\mathsf{sid}(G))\\ n=\mathsf{max}(\mathsf{pid}(G))\end{array}}{\Gamma\vdash \overline{a}[\mathtt{2..n}](\tilde{s}).P\triangleright\Delta}\right)=\mathscr{C}\langle\tilde{s},n\rangle(G)\mid\overline{a}[\mathtt{2..n,n+1}](\tilde{s},c_{\tilde{s}\mathtt{1}},\dots c_{\tilde{s}\mathtt{n}}).\mathscr{E}(\mathscr{D}_1)$$

$$\mathscr{E}\left(\text{Macc}\dfrac{\Gamma\vdash a:\langle G\rangle \quad \dfrac{\mathscr{D}_1}{\Gamma\vdash P\triangleright\Delta,\tilde{s}:(G[\mathtt{p}])@(\mathtt{p},n)} \quad |\tilde{s}|=\mathsf{max}(\mathsf{sid}(G))}{\Gamma\vdash a[\mathtt{p}](\tilde{s}).P\triangleright\Delta}\right)=a[\mathtt{p}](\tilde{s},c_{\tilde{s}\mathtt{1}},\dots c_{\tilde{s}\mathtt{n}}).\mathscr{E}(\mathscr{D}_1)$$

$$\mathscr{E}\left(\text{Branch}\dfrac{\forall l\in L:\quad \dfrac{\mathscr{D}_l}{\Gamma\vdash P_l\triangleright\Delta,\tilde{s}:T_l@(\mathtt{p},n)}}{\Gamma\vdash s_k\triangleright\{l:P_l\}_{l\in L}\triangleright\Delta,\tilde{s}:k?\{l:T_l\}_{l\in L}@(\mathtt{p},n)}\right)=s_k\triangleright\{l:c_{\tilde{s}\mathtt{p}}\triangleleft l;\mathscr{E}(\mathscr{D}_l)\}_{l\in L}$$

$$\mathscr{E}\left(\text{Sync}\dfrac{\forall l\in L'':\ \dfrac{\mathscr{D}_l}{\Gamma\vdash P_l\triangleright\Delta,\tilde{s}:T_l@(\mathtt{p},n)} \quad L''\subseteq L\cup L' \quad L'\subseteq L''}{\Gamma\vdash \mathtt{sync}_{\tilde{s},n}\{l:P_l\}_{l\in L''}\triangleright\Delta,\tilde{s}:\{l:T_l\}_{l\in L;L',n}@(\mathtt{p},n)}\right)=\begin{cases}c_{\tilde{s}\mathtt{p}}\triangleleft\mathrm{cases}_{L''};c_{\tilde{s}\mathtt{p}}\triangleright\{l:c_{\tilde{s}(\mathtt{p}\ +\mathtt{1})}\triangleleft l;\mathscr{E}(\mathscr{D}_l)\}_{l\in L''} & \text{if } \mathtt{p}\neq n\\ c_{\tilde{s}\mathtt{p}}\triangleleft\mathrm{cases}_{L''};c_{\tilde{s}\mathtt{p}}\triangleright\{l:\mathscr{E}(\mathscr{D}_l)\}_{l\in L''} & \text{if } \mathtt{p}=n\end{cases}$$

The other cases are defined monomorphic

The first case is the session request which results in a session request on the same channel. The number of participants in the new session is increased by one, to make room for the conductor process as the `n+1`-th participant, and one new session channel $c_{\tilde{s}\mathtt{p}}$ is added for each participant `p`. This channel is used by that participant to communicate with the conductor. The conductor process $\mathscr{C}\langle\tilde{s},n\rangle(G)$ is inserted in parallel with the session requesting process to make sure it is available.

The next case is the synchronisation, where we find some of the key ideas for the erasure. The resulting process starts by sending the labels it accepts to the conductor on the new session channel, and receives the chosen label on the same channel. This label is propagated through the processes, therefore when the label is received it is sent to the next process before proceeding, except when it is received by the last process. In this way the conductor only needs to send the chosen label to the first process and thereby avoiding multicast branching which is not in the system. If there are three participants in the original session as in Figure 10(c), then a synchronisation would start by the three participants sending their labels to the conductor, the conductor would send the chosen label to the first process, the first process would send the label to the second process, and the second process would send the label to the third process. The third process has no need to send the label, as all the processes have received the chosen label at this point.

Finally the case for branching results in a branching, but the receiver forwards the received label to the conductor before proceeding. This is done in order to avoid coherence problems for projection in the presence of the conductor.

Step 2: Conductor Generation The conductor process $\mathscr{C}\langle\tilde{s},n\rangle(G)$ was inserted in parallel with the session requests by the process erasure. The definition of the conductor processes $\mathscr{C}$ is in Figure 12. Notice that $\mathscr{C}\langle\tilde{s},n\rangle$ is only a wrapper for $\mathscr{C}^\star\langle\tilde{s},n\rangle$ which prefixes the session acceptance. In $\mathscr{C}\langle\tilde{s},n\rangle(G)$ the input $\tilde{s}$ is the session channels in the original session, n is the number of participants in the original session, and G is the type of the original session.

The conductor process generated from a branch receives the chosen label from the receiver of the branch, before proceeding by conducting the chosen branch. The conductor process generated from a synchronisation collects the accepted labels from each participant before sending the selected label

Figure 12 Conductor Process Generation from a Global Type

$$
\begin{array}{rcl}
\mathscr{C}\langle\tilde{s},n\rangle(G) & = & a[\mathtt{n+1}](\tilde{s},c_{\tilde{s},1},\ldots,c_{\tilde{s},n}).\mathscr{C}^\star\langle\tilde{s},n\rangle(G) \\
\mathscr{C}^\star\langle\tilde{s},n\rangle(\mathtt{p}_1\to\mathtt{p}_2:k\langle U\rangle.G') & = & \mathscr{C}^\star\langle\tilde{s},n\rangle(G') \\
\mathscr{C}^\star\langle\tilde{s},n\rangle(\mathtt{p}_1\to\mathtt{p}_2:k\{l:G_l\}_{l\in L}) & = & c_{\tilde{s},\mathtt{p}_2}\triangleright\{l:\mathscr{C}^\star\langle\tilde{s},n\rangle(G_l)\}_{l\in L} \\
\mathscr{C}^\star\langle\tilde{s},n\rangle(\{l:G_l\}_{l\in L;L'}) & = & c_{\tilde{s}\mathtt{1}}\triangleright\{\mathsf{cases}_{L_1\cup L'}:\ldots:c_{\tilde{s}\mathtt{n}}\triangleright\{\mathsf{cases}_{L_n\cup L'}: \\
 & & \mathsf{case}\ \mathsf{rand}(\bigcap_{i=1}^{n}L_i\cup L')\ \mathsf{of}\ \{l:c_{\tilde{s},1}\triangleleft l;\mathscr{C}^\star\langle n,\tilde{s}\rangle(G_l)\}_{l\in\bigcap_{i=1}^{n}L_i\cup L'}\}_{L_n\subseteq L}\ldots\}_{L_1\subseteq L} \\
\mathscr{C}^\star\langle\tilde{s},n\rangle(\mu t.G') & = & \mathsf{def}\ X_t(\tilde{s},c_{\tilde{s}\mathtt{1}},\ldots,c_{\tilde{s}\mathtt{n}})=\mathscr{C}^\star\langle\tilde{s},n\rangle(G')\ \mathsf{in}\ X_t\langle\tilde{s},c_{\tilde{s}\mathtt{1}},\ldots,c_{\tilde{s}\mathtt{n}}\rangle \\
\mathscr{C}^\star\langle\tilde{s},n\rangle(t) & = & X_t\langle\tilde{s},c_{\tilde{s}\mathtt{1}},\ldots,c_{\tilde{s}\mathtt{n}}\rangle \\
\mathscr{C}^\star\langle\tilde{s},n\rangle(\mathsf{end}) & = & \mathsf{end}
\end{array}
$$

back. The $\mathsf{case}\ \mathsf{rand}(\{l_i\}_{i\in\{1..n\}})\ \mathsf{of}\ \{l_i:P_i\}_{i\in\{1..n\}}$ constructor used in the synchronisation case of $\mathscr{C}^\star$ is syntactic sugar written with a nested conditional. The macro $\mathsf{case}\ \mathsf{rand}(\{l_i\}_{i\in\{1..n\}})\ \mathsf{of}\ \{l_i:P_i\}_{i\in\{1..n\}}$ is defined as:

$$(\nu a)(\overline{a}[2](s).s!\langle\mathsf{rand}(\{1..n\})\rangle;0\mid a[2](s).s?x.\mathsf{if}\ x=1\ \mathsf{then}\ P_1\ \mathsf{else}\ \ldots\ \mathsf{if}\ i=n-1\ \mathsf{then}\ P_{n-1}\ \mathsf{else}\ P_n)$$

Notice that the random expression is sent over a session because it is non-deterministic, and this way we can ensure that the same value is used for all the conditional branches.

Step 3: Type Translations To preserve typablity of the erasure, first we define transformations on global types, local types and message types to describe the types of the new process. This is defined in Figure 13, but we will explain some key cases.

The translation $\mathscr{G}(G)$ of global types is just a wrapper for $\mathscr{G}^\star\langle n,m\rangle(G)$ where n is the number of participants, and m is the number of session channels in the original type. The branching type is translated to the original branching type where in each branch, the receiver sends the received label to the conductor before proceeding with the selected branch. The synchronisation is translated to nested branchings, where each participant sends the accepted labels to the conductor, then the conductor sends the selected label to the first participant who forwards it to the second participant until the `n`-th participant receives the label from the `n-1`-st participant.

In the translation $\mathscr{T}\langle n,m,\mathtt{p}\rangle(T)$ of local types, n is the number of participants, m is the number of session channels and $\mathtt{p}$ is the participant of the original type T. The result of a branch type is a branch type, where the received label is sent to the conductor before proceeding with the selected branch. In the result of a synchronisation the accepted labels are sent to the conductor, then the selected label is received, and unless the process is the last process in the session, the received label is sent to the next process before proceeding.

Next we extend these transformations to the global- and local type environments in Figure 13.

Type-Preservation Theorem We will now prove that the synchronisation erasure preserves typability. The first lemma intuitively states that the generated conductor process can be typed with the communication of the `n+1`-th participant of the translated type. Therefore it has the type of the conductor.

Lemma 4.1. If $n\geq\mathsf{max}(\mathsf{pid}(G))$ and $|\tilde{s}|\geq\mathsf{max}(\mathsf{sid}(G))$ then
$\emptyset\vdash\mathscr{C}^\star\langle\tilde{s},n\rangle(G)\triangleright\tilde{s},c_{\tilde{s}\mathtt{1}},\ldots,c_{\tilde{s}\mathtt{n}}:((\mathscr{G}^\star\langle n,|\tilde{s}|\rangle(G))\upharpoonright(\mathtt{n+1}))@(\mathtt{n+1},n+1)$

Figure 13 Global, Local, Message and Environment Types Erasure Mappings

Global Type Translation

$$\mathscr{G}(G) = \mathscr{G}^\star\langle \mathsf{max}(\mathsf{pid}(G)), \mathsf{max}(\mathsf{sid}(G))\rangle(G)$$

$$\begin{array}{rcl}
\mathscr{G}^\star\langle n,m\rangle(\mathtt{p}_0 \to \mathtt{p}_1 : k\langle U\rangle.G') &=& \mathtt{p}_0 \to \mathtt{p}_1 : k\langle \mathscr{U}(U)\rangle.\mathscr{G}^\star\langle n,m\rangle(G') \\
\mathscr{G}^\star\langle n,m\rangle(\mathtt{p}_0 \to \mathtt{p}_1 : k\{l : G_l\}_{l\in L}) &=& \mathtt{p}_0 \to \mathtt{p}_1 : k\{l_j : \mathtt{p}_1 \to (\mathtt{n{+}1}) : (m+\mathtt{p}_1)\{l : \mathscr{G}^\star\langle n,m\rangle(G_l)\}\}_{l\in L} \\
\mathscr{G}^\star\langle n,m\rangle(\{l : G_l\}_{l\in L;L'}) &=& \mathtt{1} \to \mathtt{n{+}1} : (m+1)\{cases_{L_1\cup L'} : \ldots \\
& & \mathtt{n} \to \mathtt{n{+}1} : (m+n)\{cases_{L_n\cup L'} : \\
& & \mathtt{n{+}1} \to \mathtt{1} : (m+1)\{l : \\
& & \mathtt{1} \to \mathtt{2} : (m+2)\{l : \ldots \\
& & \mathtt{n{-}1} \to \mathtt{n}\{l : \mathscr{G}^\star\langle n,m\rangle(G_l)\}\ldots\}\}_{l\in\bigcap_{i=0}^{n} L_i\cup L'}\}_{L_n\subseteq L}\ldots\}_{L_1\subseteq L} \\
\mathscr{G}^\star\langle n,m\rangle(\mu t.G') &=& \mu t.\mathscr{G}^\star\langle n,m\rangle(G) \\
\mathscr{G}^\star\langle n,m\rangle(t) &=& t \\
\mathscr{G}^\star\langle n,m\rangle(\mathsf{end}) &=& \mathsf{end}
\end{array}$$

Local Type Translation

$$\begin{array}{rcl}
\mathscr{T}\langle n,m,\mathtt{p}\rangle(k!\langle U\rangle;T') &=& k!\langle \mathscr{U}(U)\rangle;\mathscr{T}\langle n,m,\mathtt{p}\rangle(T') \\
\mathscr{T}\langle n,m,\mathtt{p}\rangle(k?\langle U\rangle;T') &=& k?\langle \mathscr{U}(U)\rangle;\mathscr{T}\langle n,m,\mathtt{p}\rangle(T') \\
\mathscr{T}\langle n,m,\mathtt{p}\rangle(k\oplus\{l : T_l\}_{l\in L}) &=& k\oplus\{l : \mathscr{T}\langle n,m,\mathtt{p}\rangle(T_l)\}_{l\in L} \\
\mathscr{T}\langle n,m,\mathtt{p}\rangle(k\&\{l : T_l\}_{l\in L}) &=& k\&\{l : (m+\mathtt{p})\oplus\{l : \mathscr{T}\langle n,m,\mathtt{p}\rangle(T_l)\}\}_{l\in L} \\
\mathscr{T}\langle n,m,\mathtt{p}\rangle(\mu t.T') &=& \mu t.\mathscr{T}\langle n,m,\mathtt{p}\rangle(T') \\
\mathscr{T}\langle n,m,\mathtt{p}\rangle(t) &=& t \\
\mathscr{T}\langle n,m,\mathtt{p}\rangle(\mathsf{end}) &=& \mathsf{end} \\
\mathscr{T}\langle n,m,\mathtt{p}\rangle(\{l : T_l\}_{l\in L;L'}) &=& \\
\end{array}$$

$$\begin{cases}
(m+\mathtt{p})\oplus\{\mathrm{cases}_{L''\cup L'} : (m+\mathtt{p})\&\{case_l : (m+\mathtt{p}+1)\oplus\{case_l : \mathscr{T}\langle n,m,\mathtt{p}\rangle(T_l)\}\}_{l\in L''\cup L'}\}_{L''\subseteq L} & \text{if } \mathtt{p}\neq n \\
(m+\mathtt{p})\oplus\{\mathrm{cases}_{L''\cup L'} : (m+\mathtt{p})\&\{case_l : \mathscr{T}\langle n,m,\mathtt{p}\rangle(T_l)\}_{l\in L''\cup L'}\}_{L''\subseteq L} & \text{if } \mathtt{p}=n
\end{cases}$$

Message Type Translation

$$\begin{array}{rcl}
\mathscr{U}(\tilde{S}) &=& \tilde{S} \\
\mathscr{U}(T@(\mathtt{p},m,n)) &=& \mathscr{T}\langle n,m,\mathtt{p}\rangle(T)@(\mathtt{p},m+1)
\end{array}$$

Global Environment Translation

$$\begin{array}{rcl}
\mathscr{G}_{\mathrm{map}}(\emptyset) &=& \emptyset \\
\mathscr{G}_{\mathrm{map}}(\Gamma,u : \langle G\rangle) &=& \mathscr{G}_{\mathrm{map}}(\Gamma),u : \langle\mathscr{G}(G)\rangle \\
\mathscr{G}_{\mathrm{map}}(\Gamma,X : \tilde{S}\Delta) &=& \mathscr{G}_{\mathrm{map}}(\Gamma),X : \tilde{S}\mathscr{T}_{\mathrm{map}}(\Delta)
\end{array}$$

Local Environment Translation

$$\begin{array}{rcl}
\mathscr{T}_{\mathrm{map}}(\emptyset) &=& \emptyset \\
\mathscr{T}_{\mathrm{map}}(\Delta,\tilde{s} : T@(\mathtt{p},n)) &=& \mathscr{T}_{\mathrm{map}}(\Delta),(\tilde{s},c_{\tilde{s}1},\ldots,c_{\tilde{s}n}) : \mathscr{T}\langle|\tilde{s}|,\mathtt{p}\rangle(T)@(\mathtt{p},n+1)
\end{array}$$

PROOF: By structural induction on G. See Appendix A.2. □

The following theorem states that the conductor process can be typed with the translated type.

Theorem 4.2. If $\Gamma \vdash a : \langle G \rangle$ and $n = \mathsf{max}(\mathsf{pid}(G))$ and $|\tilde{s}| = \mathsf{max}(\mathsf{sid}(G))$ then $\mathscr{G}_{\mathrm{map}}(\Gamma) \vdash \mathscr{C}\langle \tilde{s}, n \rangle(G) \triangleright \emptyset$

PROOF: Since $\Gamma \vdash a : \langle G \rangle$ we have $\mathscr{G}_{\mathrm{map}}(\Gamma) \vdash a : \mathscr{G}(G)$. Now $\mathscr{G}(G) = \mathscr{G}^\star\langle \mathsf{max}(\mathsf{pid}(G)), \mathsf{max}(\mathsf{sid}(G)) \rangle(G)$ and therefore Lemma 4.1 gives us that $\emptyset \vdash \mathscr{C}^\star\langle \tilde{s}, n \rangle(G) \triangleright \tilde{s}, c_{\tilde{s}1}, \ldots, c_{\tilde{s}\mathtt{n}} : ((\mathscr{G}(G)) \restriction (\mathtt{n+1})) @ (\mathtt{n+1}, n+1)$ because $|\tilde{s}| = \mathsf{max}(\mathsf{sid}(G))$ and $n = \mathsf{max}(\mathsf{pid}(G))$. Then by applying Macc, we conclude the proof. □

The next lemma states that projecting the translated type is the same as translating the projected type.

Lemma 4.3. If $\mathtt{p} \leq n$, $n \geq \mathsf{max}(\mathsf{pid}(G))$ and $m \geq \mathsf{max}(\mathsf{sid}(G))$ then $\mathscr{G}^\star\langle m, n \rangle(G) \restriction \mathtt{p} = \mathscr{T}\langle m, n, \mathtt{p} \rangle(G \restriction \mathtt{p})$.

PROOF: Structural induction on G. See Appendix A.3. □

We are now ready to prove the desired theorem which states that the result of the erasure can be typed in the translated environments.

Theorem 4.4 (Type Preservation)**.** If $\dfrac{\mathscr{D}}{\Gamma \vdash P \triangleright \Delta}$ then $\mathscr{G}_{\mathrm{map}}(\Gamma) \vdash \mathscr{E}(\mathscr{D}) \triangleright \mathscr{T}_{\mathrm{map}}(\Delta)$

PROOF: Induction on the type derivation $\mathscr{D}$. See Appendix A.4.

Semantics-Preservation Theorem Next we prove that the operational semantics is preserved by the synchronisation erasure. In order to express and prove this, it is necessary to extend the erasure to runtime processes. If we extend the erasure monomorphically there would be a problem with the open sessions. This is because erasure of the starting point of a link step generates a *conductor process*, and therefore erasure of the result of a link step will have to generate the same *conductor process*. This is best explained by a small example.

Consider the process

$$P_1 = \overline{a}[2](s).\mathtt{sync}_{s,2}\{l_1 : s!\langle 1 \rangle; 0, l_2 : 0\} \mid a[2](s).\mathtt{sync}_{s,2}\{l_1 : s?(x); 0, l_2 : 0\}$$

The erasure of any type-derivation of this process will be

$$\begin{aligned} P_2 = a[3](s, c_1, c_2).C \mid \overline{a}[2,3](s, c_1, c_2).c_1 \triangleleft \mathrm{cases}_{l_1,l_2}; c_1 \triangleright \{l_1 : c_2 \triangleright l_1; s!\langle 1 \rangle; 0, l_2 : c_2 \triangleright l_2; 0\} \\ \mid a[2](s, c_1, c_2).c_2 \triangleleft \mathrm{cases}_{l_1,l_2}; c_2 \triangleright \{l_1 : s?(x); 0, l_2 : 0\} \end{aligned}$$

for some conductor C. When the process P_2 steps we get

$$\begin{aligned} P_2' = (\nu \tilde{s}, c_1, c_2)(C \mid c_1 \triangleleft \mathrm{cases}_{l_1,l_2}; c_1 \triangleright \{l_1 : c_2 \triangleright l_1; s!\langle 1 \rangle; 0, l_2 : c_2 \triangleright l_2; 0\} \\ \mid c_2 \triangleleft \mathrm{cases}_{l_1,l_2}; c_2 \triangleright \{l_1 : s?(x); 0, l_2 : 0\}) \end{aligned}$$

However, if we instead start by stepping from P_1 we get

$$P_1' = (\nu \tilde{s})(\mathtt{sync}_{s,2}\{l_1 : s!\langle 1 \rangle; 0, l_2 : 0\} \mid \mathtt{sync}_{s,2}\{l_1 : s?(x); 0, l_2 : 0\})$$

and if the erasure is extended monomorphically the erasure of any type-derivation of P_1' will be

$$\begin{aligned} P_2'' = (\nu \tilde{s})(c_1 \triangleleft \mathrm{cases}_{l_1,l_2}; c_1 \triangleright \{l_1 : c_2 \triangleright l_1; s!\langle 1 \rangle; 0, l_2 : c_2 \triangleright l_2; 0\} \\ \mid c_2 \triangleleft \mathrm{cases}_{l_1,l_2}; c_2 \triangleright \{l_1 : s?(x); 0, l_2 : 0\}) \end{aligned}$$

Hence the conductor process is missing. Therefore we need to recover the *partial constructors* from the local types when session channels are hidden to show an operational correspondence.

We solve this by extending the erasure to runtime processes monomorphically except for the rule CRes where the erasure is defined in the following way

$$\mathscr{E}\left(\text{CRes}\frac{\dfrac{\mathscr{D}_1}{\Gamma\vdash P\rhd_{\tilde{t}}\Delta,\tilde{s}:\{G|\mathtt{p}@(\mathtt{p},n)\}_{\mathtt{p}\in\{1..n\}}}\quad \tilde{s}\subseteq\tilde{t}\quad G\text{ coherent}}{\Gamma\vdash(\nu\tilde{s})P\rhd_{\tilde{t}\setminus\tilde{s}}\Delta}\right)$$
$$=(\nu\tilde{s},c_{\tilde{s}1},\ldots,c_{\tilde{s}\mathtt{n}})(\mathscr{C}^\star\langle\tilde{s},n\rangle(G)\mid c_{\tilde{s}1}:\emptyset\mid\ldots\mid c_{\tilde{s}\mathtt{n}}:\emptyset\mid\mathscr{E}(\mathscr{D}_1))$$

With this definition, the semantics is preserved by the erasure, when $\Delta=\emptyset$, but in order to prove the theorem we have to generalise it in the following way. In the Cres rule, the session types are moved from Δ to the syntax, and therefore there is a problem using the induction hypothesis in the Scop case. In order to solve this we define PC as the set of possible partial conductor processes generated from Δ, and generalise the theorem to include these processes. This only works when Δ holds *all* the projections of the global type, and therefore we need the following definitions.

$\emptyset$ complete, and
Δ complete implies $\Delta,\tilde{s}:\{T_\mathtt{p}@(\mathtt{p},n)\}_{\mathtt{p}\in\{1..n\}}$ complete.

$\mathtt{PC}(\emptyset)=\{0\}$
$\mathtt{PC}(\Delta',\tilde{s}:\{T_\mathtt{p}@(\mathtt{p},n)\}_{\mathtt{p}\in\{1..n\}})$
$\quad=\bigcup_{P'_C\in\mathtt{PC}(\Delta')}\{\mathscr{C}^\star\langle\tilde{s},n\rangle(G)|P'_C|c_{\tilde{s}1}:\emptyset|\ldots|c_{\tilde{s}\mathtt{n}}:\emptyset\mid G|\mathtt{p}=T_\mathtt{p}\ \forall\mathtt{p}\in\{1..n\}\}$

Now we are ready to express the semantics preservation theorem in a provable way.

Theorem 4.5.
If $\dfrac{\mathscr{D}}{\Gamma\vdash P\rhd_{\tilde{t}}\Delta}$, $P\to P'$, Δ coherent, $\Delta\circ\Delta^\circ$ complete and $P_C\in\mathtt{PC}(\Delta\circ\Delta^\circ)$
then there is a derivation $\dfrac{\mathscr{D}'}{\Gamma\vdash P'\rhd_{\tilde{t}}\Delta'}$ and a $P'_C\in\mathtt{PC}(\Delta'\circ\Delta^\circ)$ where $\Delta=\Delta'$ or $\Delta\to\Delta'$
such that $\mathscr{E}(\mathscr{D})|P_C\to^\star\mathscr{E}(\mathscr{D}')|P'_C$.
PROOF: Induction on the derivation of $P\to P'$. See Appendix A.7.

Now the semantic preservation is proved by applying Theorem 4.5 to each step in an evaluation.

Corollary 4.6 (Semantic Preservation)**.**
If $\dfrac{\mathscr{D}}{\Gamma\vdash P\rhd\emptyset}$ and $P\to^\star P'$ then there is a derivation $\dfrac{\mathscr{D}'}{\Gamma\vdash P'\rhd\emptyset}$ such that $\mathscr{E}(\mathscr{D})\to^\star\mathscr{E}(\mathscr{D}')$.
PROOF: Induction on number of steps in $P\to^\star P'$. See Appendix A.8.

Healthcare Cooperation Example In this section we have demonstrated how we can translate a well-typed process using the `sync` constructor to a well-typed process with no occurrences of `sync` in a type-directed manner. The translation preserves typability, following the translation of types. As a example, the result of the erasure on the healthcare example from Section 3 is shown in Figure 14.

Proposition 4.7 (Example is well-typed)**.** $a:\langle\mathscr{G}(G)\rangle\vdash P_C\mid P_D\mid P_N\mid P_P\rhd\emptyset$.
Since we have showed that the processes from the synchronisation example in Figure 5 is well-typed in Proposition 3.1 we can apply Theorem 4.4 which proves this proposition. □

As this example illustrates, the result of the erasure does not capture the nature of the situation in the same way, as it introduces a conductor process, which is not a natural part of the situation. It is not compact either, as the conductor process has 8^3 cases. This can be improved by using a less straightforward erasure algorithm, but it will still not be as compact as the representation using `sync` — removing the `sync`s is at the cost of a natural and direct representation of such a cooperative action.

Figure 14 Example Processes after Erasure

$$
\begin{array}{l}
P_C = a[4](p1,p2,d,n,cp,cd,cn).\\
\quad cp \triangleright \{\\
\quad\quad \text{cases}_{\{1,2,3,4\}} : cd \triangleright \{\\
\quad\quad\quad\quad \text{cases}_{\{1,2,3,4\}} : cn \triangleright \{\\
\quad\quad\quad\quad\quad\quad \text{cases}_{\{1,2,3,4\}} :\\
\quad\quad\quad\quad\quad\quad \mathsf{case}\ (\mathsf{rand}\{1,2,3,4\})\ \mathsf{of}\\
\quad\quad\quad\quad\quad\quad\quad 1 : cp \triangleleft \text{case1}; 0,\\
\quad\quad\quad\quad\quad\quad\quad 2 : cp \triangleleft \text{case2}; 0,\\
\quad\quad\quad\quad\quad\quad\quad 3 : cp \triangleleft \text{case3}; 0,\\
\quad\quad\quad\quad\quad\quad\quad 4 : cp \triangleleft \text{case4}; 0,\\
\quad\quad\quad\quad\quad\quad \mathsf{end\ case},\\
\quad\quad\quad\quad\quad\quad \text{cases}_{\{2,3,4\}} : ...,\\
\quad\quad\quad\quad\quad\quad ...,\\
\quad\quad\quad\quad\quad\quad \text{cases}_{\{2\}} : ...\}\\
\quad\quad\quad\quad \text{cases}_{\{2,3,4\}} : ...,\\
\quad\quad\quad\quad ...,\\
\quad\quad\quad\quad \text{cases}_{\{2\}} : ...\}\\
\quad\quad \text{cases}_{\{2,3,4\}} : ...,\\
\quad\quad ...,\\
\quad\quad \text{cases}_{\{2\}} : ...\}
\end{array}
$$

$$
\begin{array}{l}
P_D = a[2](p1,p2,d,n,cp,cd,cn).cd \triangleleft \text{cases}_{\{1,2,3,4\}}.cd \triangleright \{\\
\quad \text{case1} : cn \triangleleft case1; d?(\text{data}); p1!\langle e_{\text{schedule}}\rangle; p1!\langle e_{\text{result}}\rangle; 0\\
\quad \text{case2} : cn \triangleleft case2; p1!\langle e_{\text{schedule}}\rangle; p1!\langle e_{\text{result}}\rangle; 0\\
\quad \text{case3} : cn \triangleleft case3; d?(\text{data}); p1!\langle e_{\text{result}}\rangle; 0\\
\quad \text{case4} : cn \triangleleft case4; p1!\langle e_{\text{result}}\rangle; 0\}\\
\\
P_N = a[3](p1,p2,d,n,cp,cd,cn).cn \triangleleft \text{cases}_{\{1,2,3,4\}}.cn \triangleright \{\\
\quad \text{case1} : 0\\
\quad \text{case2} : n?(\text{data}); 0\\
\quad \text{case3} : p2!\langle e_{\text{schedule}}\rangle; 0\\
\quad \text{case4} : n?(\text{data}); p2!\langle e_{\text{schedule}}\rangle; 0\}\\
\\
P_P = \overline{a}[2,3,4](p1,p2,d,n,cp,cd,cn).cp \triangleleft \text{cases}_{\{1,2,3,4\}}.cp \triangleright \{\\
\quad \text{case1} : cd \triangleleft \text{case1}; d!\langle e_{\text{data}}\rangle; p1?(\text{schedule}); p1?(\text{result}); 0\\
\quad \text{case2} : cd \triangleleft \text{case2}; n!\langle e_{\text{data}}\rangle; p1?(\text{schedule}); p1?(\text{result}); 0\\
\quad \text{case3} : cd \triangleleft \text{case3}; d!\langle e_{\text{data}}\rangle; p2?(\text{schedule}).p1?(\text{result}); 0\\
\quad \text{case4} : cd \triangleleft \text{case4}; n!\langle e_{\text{data}}\rangle; p2?(\text{schedule}).p1?(\text{result}); 0\}
\end{array}
$$

Further we lose an accurate type abstraction of the dynamics of symmetric synchronisation, since from the encoded type structure, it is not clear whether it is just a sequence of asymmetric branching actions or the (intended) atomic multiparty synchronisation, since some of the key operational structures of the encoding (e.g. random selection) is lost in the encoded type. On the other hand, type-directed translation suggests the existence of a well-disciplined implementation of symmetric synchronisation through the standard asynchronous, asymmetric two-party communication primitives.

5 Future Work

The short term goal of this paper is to provide a basis for representation and specification of the *Clinical Practice Guidelines* [11] modelled by the *Process Matrix* [12] using the π-calculus. The long term goals are to allow representation and specification of a wider range of Clinical Practice Guidelines and other Healthcare Processes, and to model a wide range of *structured social interactions* [1] in the π-calculus.

In the present study, we have chosen to ensure progress by making the type system force the participants to agree in each synchronisation, so they always share a common label. Another possibility is to allow the option for *disagreement*, so no labels are accepted. This would mean that the synchronisation is unable to step. Therefore it will be necessary to introduce some kind of *disagreement handling* unless progress is sacrificed. This approach can be represented by the current extension simply by introducing a partial order on the labels, and force the synchronisation to proceed with a *maximal* element of the shared labels. In this way a disagreement would be handled if only the *bottom element* label was shared.

Another extention is to formalise synchronisation between any *subset* of the participants in a session. This would introduce new linearity and coherence problems which would have to be dealt with.

Finally using `sync` we can avoid the linearity check in [5] in the following way. Consider the process:

$$s!\langle e\rangle; 0 \mid s?(x); s?(y); 0 \mid s!\langle e\rangle; 0.$$

This process does not satisfy the linearity because there is a racing condition at s. To avoid this problem [5] uses the linearity checking for global types. However, interestingly, we can avoid this directly by inserting `sync` primitives between the two messages like this:

$$s!\langle e\rangle;\mathtt{sync}_{s,3}\{l:0\} \mid s?(x);\mathtt{sync}_{s,3}\{l:s?(y);0\} \mid \mathtt{sync}_{s,3}\{l:s!\langle e\rangle;0\}.$$

Now the racing condition has been eliminated. It should be investigated how this method can avoid racing conditions so the linearity check can be removed or relaxed.

References

[1] *Wikipedia on Social Interactoins*. Available at `http://en.wikipedia.org/wiki/Social_interaction`.

[2] Eduardo Bonelli & Adriana B. Compagnoni (2007): *Multipoint Session Types for a Distributed Calculus*. In: *TGC*, *LNCS* 4912. Springer, pp. 240–256.

[3] Tony Hoare (1985): *Communicating Sequential Processes*. Prentice Hall.

[4] Kohei Honda, Vasco T. Vasconcelos & Makoto Kubo (1998): *Language Primitives and Type Disciplines for Structured Communication-based Programming*. In: *ESOP'98*, *LNCS* 1381. Springer-Verlag, pp. 22–138.

[5] Kohei Honda, Nobuko Yoshida & Marco Carbone (2008): *Multiparty Asynchronous Session Types*. In: *POPL'08*. ACM, pp. 273–284.

[6] Uwe Nestmann (2000): *What is a "Good" Encoding of Guarded Choice?* *Inf. Comput.* 156(1-2), pp. 287–319.

[7] Uwe Nestmann & Benjamin C. Pierce (2000): *Decoding Choice Encodings*. *Inf. Comput.* 163(1), pp. 1–59.

[8] Catuscia Palamidessi (2003): *Comparing The Expressive Power Of The Synchronous And Asynchronous Pi-Calculi*. *Mathematical Structures in Computer Science* 13(5), pp. 685–719.

[9] K.V.S. Prasad (2001): *Broadcast Calculus Interpreted in CCS upto Bisimulation*. In: *Electronic Notes in Theoretical Computer Science*, 52. Elsevier, pp. 83–100.

[10] Kaku Takeuchi, Kohei Honda & Makoto Kubo (1994): *An Interaction-based Language and its Typing System*. In: *PARLE'94*, *LNCS* 817. Springer-Verlag, pp. 398–413.

[11] Silvia Miksch Annette ten Teije & Peter Lucas (2008): *Computer-based Medical Guidelines and Protocols: A Primer and Currend Trends*. Studies in Health Technology and Informatics. IOS Press.

[12] Karen Marie Lyng Thomas Hildebrandt & Raghava Rao Mukkamala (2009): *From Paper Based Clinical Practice Guidelines to Declarative Workflow and Linear-time Temporal Logic* .

A Appendix

Figure 15 Process Congruence

$P|0 \equiv P$ $\quad P|Q \equiv Q|P$ $\quad P|(Q|R) \equiv (P|Q)|R$
$(\nu n)P|Q \equiv (\nu n)(P|Q)$ if $n \notin \mathsf{fn}(Q)$ $\quad (\nu nn')P \equiv (\nu n'n)P$ $\quad (\nu n)0 \equiv 0$
$\mathsf{def}\ D\ \mathsf{in}\ 0 \equiv 0$ $\quad (\nu s_1..s_n)\Pi_i s_i : \emptyset \equiv 0$
$\mathsf{def}\ D\ \mathsf{in}\ (\nu n)P \equiv (\nu n)\mathsf{def}\ D\ \mathsf{in}\ P$ if $n \notin \mathsf{fn}(D)$
$(\mathsf{def}\ D\ \mathsf{in}\ P) \mid Q \equiv \mathsf{def}\ D\ \mathsf{in}\ (P \mid Q)$ if $\mathsf{dpv}(D) \cap \mathsf{fpv}(Q) = \emptyset$
$\mathsf{def}\ D\ \mathsf{in}\ \mathsf{def}\ D'\ \mathsf{in}\ P \equiv \mathsf{def}\ D\ \mathsf{and}\ D'\ \mathsf{in}\ P$ if $\mathsf{dpv}(D) \cap \mathsf{dpv}(D') = \emptyset$

Figure 16 Subtyping for Local Types

The subtyping for local types from [5, § 5], denoted $T \leq_{\text{sub}} T'$ is the *greatest fixed point* of the function S that maps each binary relation R on local types as regular trees to $S(R)$ given as

If TRT' then $k!\langle U\rangle;T\ S(R)\ k!\langle U\rangle;T'$ and $k?\langle U\rangle;T\ S(R)\ k?\langle U\rangle;T$
If T_iRT_i', for each $i \in I \subseteq J$ then $k \oplus \{l_i : T_i\}_{i\in I}\ S(R)\ k \oplus \{l_j : T_j'\}_{j\in J}$ and $k\&\{l_j : T_j\}_{j\in J}\ S(R)\ k\&\{l_i : T_i'\}_{i\in I}$
and $\{l_i : T_i\}_{i\in I}\ S(R)\ \{l_i : T_i'\}_{i\in I}$

A.1 Proof of Theorem 3.2[Subject Reduction]

We prove

$\Gamma \vdash P \triangleright_{\tilde{s}} \Delta$ and Δ coherent and $P \to P'$ implies $\Gamma \vdash P' \triangleright_{\tilde{s}} \Delta'$where $\Delta = \Delta'$ or $\Delta \to \Delta'$.

By induction on the derivation of $P \to P'$. We only have to consider the case Sync, as the other cases are proved in [5]. Assume

$$\text{Sync}\frac{h \in \bigcap_{i=1}^{n} L_i}{\mathtt{sync}_{\tilde{s},n}\{l : P_{1l}\}_{l\in L_1} \mid ... \mid \mathtt{sync}_{\tilde{s},n}\{l : P_{nl}\}_{l\in L_n} \to P_{1h} \mid ... \mid P_{nh}}$$

In this case the typing $\Gamma \vdash \mathtt{sync}_{\tilde{t},n}\{l : P_{1l}\}_{l\in L_1} \mid ... \mid \mathtt{sync}_{\tilde{t},n}\{l : P_{nl}\}_{l\in L_n} \triangleright_{\tilde{s}} \Delta, \tilde{t} : \{\{l : T_{li}\}_{l\in L;L'}@(\mathtt{i},n)\}_{i\in\{1..n\}}$ must start with $n-1$ applications of the Conc rule each containing one application of the Sync rule. This gives us the subderivations:

$$\text{Sync}\frac{\dfrac{\mathscr{D}_{il}}{\Gamma \vdash P_{il} \triangleright_{\tilde{s}_i} \Delta_i, \tilde{t} : \{T_{li}@(\mathtt{i},n)\}} \quad \forall l \in L_i}{\Gamma \vdash \{l : P_{il}\}_{l\in L_i} \triangleright_{\tilde{s}_i} \Delta_i, \tilde{t} : \{l : T_{li}\}_{l\in L';L}@(\mathtt{i},n)}$$

for i=1..n such that $\tilde{s}_i \cap \tilde{s}_j = \emptyset$ for all $i \neq j$ in $1..n$, $\bigcup_{i=1}^{n} \tilde{s}_i = \tilde{s}$ and $\Delta_1 \circ (\Delta_2 \circ (\ldots \circ \Delta_n)) = \Delta$.

Since each of these subderivations starts with the Sync rule we get that

$$\frac{\mathscr{D}_{ih}}{\Gamma \vdash P_{ih} \triangleright_{\tilde{s}_i} \Delta_i, \tilde{t} : \{T_{hi}@(\mathtt{i},n)\}} \text{ for } i = 1..n$$

Now we can apply Conc $n-1$ times to create a derivation of $\Gamma \vdash P_{1h} \mid \ldots \mid P_{nh} \triangleright_{\tilde{s}} \Delta, \tilde{t} : \{T_{hi}@(\mathtt{p},n)\}_{\mathtt{p}\in\{1..n\}}$. Then we have $\Delta, \tilde{t} : \{\{l : T_{li}\}_{l\in L;L'}@(\mathtt{i},n)\}_{i\in\{1..n\}} \to \Delta, \tilde{t} : \{T_{hi}@(\mathtt{i},n)\}_{i\in\{1..n\}}$, concluding the proof for the Sync rule. □

Figure 17 The typing rules

$$\text{Name}\frac{}{\Gamma, a : S \vdash a : S} \qquad \text{Subs}\frac{\Gamma \vdash P \triangleright \Delta \quad \Delta \leq \Delta'}{\Gamma \vdash P \triangleright \Delta'}$$

$$\text{Sync}\frac{\forall l \in L'' : \Gamma \vdash P_l \triangleright \Delta, \tilde{s} : T_l @ (\mathtt{p}, n) \quad L'' \subseteq L \cup L' \quad L' \subseteq L''}{\Gamma \vdash \mathtt{sync}_{\tilde{s},n}\{l : P_l\}_{l \in L''} \triangleright \Delta, \tilde{s} : \{l : T_l\}_{l \in L;L'} @ (\mathtt{p}, n)}$$

$$\text{Mcast}\frac{\Gamma \vdash a : \langle G \rangle \quad \Gamma \vdash P \triangleright \Delta, \tilde{s} : (G \upharpoonright 1) @ (1, n) \quad |\tilde{s}| = \mathsf{max}(\mathsf{sid}(G)) \quad n = \mathsf{max}(\mathsf{pid}(G))}{\Gamma \vdash \overline{a}[2..\mathtt{n}](\tilde{s}).P \triangleright \Delta}$$

$$\text{Macc}\frac{\Gamma \vdash a : \langle G \rangle \quad \Gamma \vdash P \triangleright \Delta, \tilde{s} : (G \upharpoonright \mathtt{p}) @ (\mathtt{p}, n) \quad |\tilde{s}| = \mathsf{max}(\mathsf{sid}(G)) \quad n = \mathsf{max}(\mathsf{pid}(G))}{\Gamma \vdash a[\mathtt{p}](\tilde{s}).P \triangleright \Delta}$$

$$\text{Send}\frac{\forall j . \Gamma \vdash e_j : S_j \quad \Gamma \vdash P \triangleright \Delta, s : T @ (\mathtt{p}, n)}{\Gamma \vdash s_k ! \langle \tilde{e} \rangle ; P \triangleright \Delta, \tilde{s} : k ! \langle \tilde{S} \rangle ; T @ (\mathtt{p}, n)}$$

$$\text{Rcv}\frac{\Gamma, \tilde{x} : \tilde{S} \vdash P \triangleright \Delta, s : T @ (\mathtt{p}, n)}{\Gamma \vdash s_k ? (\tilde{x}) ; P \triangleright \Delta, \tilde{s} : k ? \langle \tilde{S} \rangle ; T @ (\mathtt{p}, n)}$$

$$\text{Deleg}\frac{\Gamma \vdash P \triangleright \Delta, \tilde{s} : T @ (\mathtt{p}, n)}{\Gamma \vdash s_k ! \langle \langle \tilde{t} \rangle \rangle ; P \triangleright \Delta, \tilde{s} : k ! \langle T' @ (\mathtt{p}', |\tilde{t}|, n') \rangle ; T @ (\mathtt{p}, n), \tilde{t} : T' @ (\mathtt{p}', n')}$$

$$\text{Srec}\frac{\Gamma \vdash P \triangleright \Delta, \tilde{s} : T @ (\mathtt{p}, n), \tilde{t} : T' @ (\mathtt{p}', n')}{\Gamma \vdash s_k ? ((\tilde{t})) ; P \triangleright \Delta, \tilde{s} : k ? \langle T' @ (\mathtt{p}', |\tilde{t}|, n') \rangle ; T @ (\mathtt{p}, n)} \qquad \text{Sel}\frac{\Gamma \vdash P \triangleright \Delta, \tilde{s} : T @ (\mathtt{p}, n) \quad h \in L}{\Gamma \vdash s_k \triangleleft h ; P \triangleright \Delta, \tilde{s} : k \oplus \{l : T_l\}_{l \in L} @ (\mathtt{p}, n)}$$

$$\text{Branch}\frac{\forall l \in L : \quad \Gamma \vdash P_l \triangleright \Delta, \tilde{s} : T_l @ (\mathtt{p}, n)}{\Gamma \vdash s_k \triangleright \{l : P_l\}_{l \in L} \triangleright \Delta, \tilde{s} : k ? \{l : T_l\}_{l \in L} @ (\mathtt{p}, n)}$$

$$\text{Conc}\frac{\Gamma \vdash P \triangleright \Delta \quad \Gamma \vdash Q \triangleright \Delta'}{\Gamma \vdash P | Q \triangleright \Delta \circ \Delta'} \ \mathsf{dom}(\Delta) \cap \mathsf{dom}(\Delta') = \emptyset$$

$$\text{If}\frac{\Gamma \vdash e : \mathsf{bool} \quad \Gamma \vdash P \triangleright \Delta}{\Gamma \vdash \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \triangleright \Delta} \qquad \text{Inact}\frac{\Delta \quad \mathsf{end\ only}}{\Gamma \vdash 0 \triangleright \Delta} \qquad \text{Nres}\frac{\Gamma, a : \langle G \rangle \vdash P \triangleright \Delta}{\Gamma \vdash (\nu a) P \triangleright \Delta}$$

$$\text{Var}\frac{\Gamma \vdash \tilde{e} : \tilde{S} \quad \Delta \quad \mathsf{end\ only}}{\Gamma, X : \tilde{S}\tilde{T} \vdash X \langle \tilde{e} \tilde{s}_1 \ldots \tilde{s}_{|\tilde{T}|} \rangle \triangleright \Delta, \tilde{s}_1 : T_1 @ (\mathtt{p}_1, n_1), \ldots, \tilde{s}_n : T_{|\tilde{T}|} @ (\mathtt{p}_{|\tilde{T}|}, n_{|\tilde{T}|})}$$

$$\text{Def}\frac{\Gamma, X : \tilde{S}\tilde{T}, \tilde{x} : \tilde{S} \vdash P \triangleright \tilde{s}_1 : T_1 @ (\mathtt{p}_1, n_1), \ldots, \tilde{s}_{|\tilde{T}|} : T_{|\tilde{T}|} @ (\mathtt{p}_{|\tilde{T}|}, n_{|\tilde{T}|}) \quad \Gamma, X : \tilde{S}\tilde{T} \vdash Q \triangleright \Delta}{\Gamma \vdash \mathsf{def}\ X(\tilde{x}\tilde{s}_1, \ldots, \tilde{s}_{|\tilde{T}|}) = P\ \mathsf{in}\ Q \triangleright \Delta}$$

A.2 Proof of Lemma 4.1

We prove

If $n \geq \mathsf{max}(\mathsf{pid}(G))$ and $|\tilde{s}| \geq \mathsf{max}(\mathsf{sid}(G))$ then
$\emptyset \vdash \mathscr{C}^\star \langle \tilde{s}, n \rangle (G) \triangleright \tilde{s}, c_{\tilde{s}1}, \ldots, c_{\tilde{s}\mathtt{n}} : ((\mathscr{G}^\star \langle n, |\tilde{s}| \rangle (G)) \upharpoonright (\mathtt{n+1})) @ (\mathtt{n+1}, n+1)$

By structural induction on G. The interesting cases Branch and Sync are explained briefly.

Branch: Assume $G = \mathtt{p}_1 \to \mathtt{p}_2 : k\{l : G_l\}_{l \in L}$.
Then the resulting conductor process is: $\mathscr{C}^\star \langle \tilde{s}, n \rangle (G) = c_{\tilde{s}\mathtt{p}_2} \triangleright \{l : \mathscr{C}^\star \langle \tilde{s}, n \rangle (G_l)\}_{l \in L}$, and the global type is: $\mathscr{G}^\star \langle n, |\tilde{s}| \rangle (G) = \mathtt{p}_1 \to \mathtt{p}_2 : k\{l : \mathtt{p}_2 \to \mathtt{n+1} : c_{\tilde{s}\mathtt{p}_2}\{l : \mathscr{G}^\star \langle n, |\tilde{s}| \rangle (G_l)\}\}_{l \in L}$. Therefore the resulting local type is: $(\mathscr{G}^\star \langle n, |\tilde{s}| \rangle (G)) \upharpoonright (\mathtt{n+1}) = \max_{\leq_{\mathsf{sub}}}\{T' \mid \forall l \in L . T' \leq_{\mathtt{sub}} (|\tilde{s}| + \mathtt{p}_2) \& \{l : \mathscr{G}^\star \langle n, |\tilde{s}| \rangle (G_l)\}\} = (|\tilde{s}| + \mathtt{p}_2) \& \{l : \mathscr{G}^\star \langle n, |\tilde{s}| \rangle (G_l)\}_{l \in L}$ with $\mathtt{n+1} > \mathtt{p}_1$ and $\mathtt{p}_2$.

By the induction hypothesis, $\emptyset \vdash \mathscr{C}^\star\langle \tilde{s}, n\rangle(G_l) \triangleright \tilde{s}, c_{\tilde{s}1}, \ldots, c_{\tilde{s}\mathtt{n}} : (\mathscr{G}^\star\langle n, |\tilde{s}|\rangle(G_l)\upharpoonright(\mathtt{n+1}))@(\mathtt{n+1}, n+1)$ for all $l \in L$, and because of rule Branch we get that $\emptyset \vdash c_{\tilde{s}\mathtt{p}_2} \triangleright \{l : \mathscr{C}^\star\langle \tilde{s}, n\rangle(G_l)\}_{l\in L} \triangleright \tilde{s}, c_{\tilde{s}1}, \ldots, c_{\tilde{s}\mathtt{n}} : ((|\tilde{s}| + \mathtt{p}_2)\&\{\mathscr{G}^\star\langle n, |\tilde{s}|\rangle(G_l)\upharpoonright(\mathtt{n+1})\}_{l\in L})@(\mathtt{n+1}, n+1)$. Therefore this case is fulfilled.

Sync: Assume $G = \{l : G_l\}_{l\in L;L'}$. In this case the resulting conductor process is: $\mathscr{C}^\star\langle \tilde{s}, n\rangle(G) = c_{\tilde{s}1} \triangleright \{\mathrm{cases}_{L_1''\cup L'} : c_{\tilde{s}2} \triangleright \{\mathrm{cases}_{L_2''\cup L'} : \ldots c_{\tilde{s}\mathtt{n}} \triangleright \{\mathrm{cases}_{L_n''\cup L'} : \mathsf{case}\ \mathsf{rand}(L' \cup (L_1'' \cap L_2'' \cap \ldots L_n''))\ \mathsf{of}\ \{l : c_{\tilde{s}1} \triangleleft l; \mathscr{C}^\star\langle \tilde{s}, n\rangle(G_l)\}_{l\in L'\cup(L_1''\cap L_2''\cap\ldots L_n'')}\}_{L_n''\subseteq L}\ldots\}_{L_2''\subseteq L}\}_{L_1''\subseteq L}$, the resulting global type is

$$\begin{aligned}\mathscr{G}^\star\langle n, |\tilde{s}|\rangle(G) = \mathtt{1} \to \mathtt{n+1} : (|\tilde{s}|+1)\{&\mathrm{cases}_{L_1''\cup L'} : \mathtt{2} \to \mathtt{n+1} : (|\tilde{s}|+2)\{\mathrm{cases}_{L_2''\cup L'} : \ldots\\ &\mathtt{n} \to \mathtt{n+1} : (|\tilde{s}|+\mathtt{n})\{\mathrm{cases}_{L_n''\cup L'} : \mathtt{n+1} \to \mathtt{1} : (|\tilde{s}|+1)\{l : \mathtt{1} \to \mathtt{2} : (|\tilde{s}|+2)\{l : \ldots\\ &\mathtt{n-1} \to \mathtt{n} : (|\tilde{s}|+\mathtt{n})\{l : \mathscr{G}^\star\langle n, |\tilde{s}|\rangle(G_l)\}\}\}_{l\in L'\cup(L_1''\cap L_2''\cap\ldots L_n'')}\}_{L_n''\subseteq L}\ldots\}_{L_2''\subseteq L}\}_{L_1''\subseteq L}\end{aligned}$$

and therefore the resulting local type is

$$\begin{aligned}(\mathscr{G}^\star\langle n, |\tilde{s}|\rangle(G))\upharpoonright(\mathtt{n+1}) = (|\tilde{s}|+1)\&\{&\mathrm{cases}_{L_1''\cup L'} : (|\tilde{s}|+2)\&\{\mathrm{cases}_{L_2''\cup L'} : \ldots (|\tilde{s}|+\mathtt{n})\&\{\mathrm{cases}_{L_n''\cup L'} :\\ &(|\tilde{s}|+1) \oplus \{l : \mathscr{G}^\star\langle n, |\tilde{s}|\rangle(G_l)\upharpoonright(\mathtt{n+1})\}_{l\in L'\cup(L_1''\cap L_2''\cap\ldots L_n'')}\}_{L_n''\subseteq L}\ldots\}_{L_2''\subseteq L}\}_{L_1''\subseteq L},\end{aligned}$$

since $\mathtt{n+1} > \mathtt{p}_1$ and $\mathtt{p}_2$.

We get from the induction hypothesis that $\dfrac{\mathscr{D}_l}{\emptyset \vdash \mathscr{C}^\star\langle \tilde{s}, n\rangle(G_l) \triangleright \tilde{s}, c_{\tilde{s}1}, \ldots, c_{\tilde{s}\mathtt{n}} : (\mathscr{G}^\star\langle n, |\tilde{s}|\rangle(G_l)\upharpoonright(\mathtt{n+1}))@(\mathtt{n+1}, n+1)}$ for all $l \in L \cup L'$. We can therefore construct the type derivation in by n levels of Branch rules on top of which are nested If rules with a single Sel rule containing $\mathscr{D}_l$ for the selected branch l. Therefor this case is fulfilled.

We have now proved the interesting cases therefore the lemma is fulfilled. □

A.3 Proof of Lemma 4.3

We prove

$$\text{If } \mathtt{p} \le n \text{ and } n \ge \mathsf{max}(\mathsf{pid}(G)) \text{ and } m \ge \mathsf{max}(\mathsf{sid}(G)) \text{ then } \mathscr{G}^\star\langle m, n\rangle(G)\upharpoonright\mathtt{p} = \mathscr{T}\langle m, n, \mathtt{p}\rangle(G\upharpoonright\mathtt{p}).$$

By the structural induction on G. The interesting cases Branch and Sync are explained briefly.

Branch: $G = \mathtt{p}_0 \to \mathtt{p}_1 : k\{l : G_l\}_{l\in L}$

There are three subcases depending on $\mathtt{p}$.

If $\mathtt{p} \neq \mathtt{p}_0$ and $\mathtt{p} \neq \mathtt{p}_1$ then

$$\begin{aligned}&\mathscr{G}^\star\langle n, m\rangle(\mathtt{p}_0 \to \mathtt{p}_1 : k\{l : G_l\}_{l\in L})\upharpoonright\mathtt{p}\\ &\quad= (\mathtt{p}_0 \to \mathtt{p}_1 : k\{l : \mathtt{p}_1 \to \mathtt{n+1} : m + \mathtt{p}_1 : \{l : \mathscr{G}^\star\langle n, m\rangle(G_l)\}\}_{l\in L})\upharpoonright\mathtt{p}\\ &\quad= \max_{\le_{\mathrm{sub}}}\{T \mid T \le_{\mathrm{sub}} \mathtt{p}_1 \to \mathtt{n+1} : m + \mathtt{p}_1\{l : \mathscr{G}^\star\langle n, m\rangle(G_l)\}\upharpoonright\mathtt{p} \quad \forall l \in L\}\\ &\quad= \max_{\le_{\mathrm{sub}}}\{T \mid T \le_{\mathrm{sub}} \max_{\le_{\mathrm{sub}}}\{T' \mid T' \le_{\mathrm{sub}} \mathscr{G}^\star\langle n, m\rangle(G_l)\upharpoonright\mathtt{p}\} \quad \forall l \in L\}\\ &\quad= \max_{\le_{\mathrm{sub}}}\{T \mid T \le_{\mathrm{sub}} \mathscr{G}^\star\langle n, m\rangle(G_l)\upharpoonright\mathtt{p} \quad \forall l \in L\}\end{aligned}$$

and

$$\begin{aligned}&\mathscr{T}\langle n, m, \mathtt{p}\rangle(\mathtt{p}_0 \to \mathtt{p}_1 : k\{l : G_l\}_{l\in L}\upharpoonright\mathtt{p})\\ &\qquad= \mathscr{T}\langle n, m, \mathtt{p}\rangle(\max_{\le_{\mathrm{sub}}}\{T \mid T \le_{\mathrm{sub}} G_l\upharpoonright\mathtt{p} \quad \forall l \in L\})\\ &(\mathscr{T} \text{ is monotonic}) = \max_{\le_{\mathrm{sub}}}\{T \mid T \le_{\mathrm{sub}} \mathscr{T}\langle n, m, \mathtt{p}\rangle(G_l\upharpoonright\mathtt{p}) \quad \forall l \in L\}\end{aligned}$$

Now the induction hypothesis proves this case.

The cases where $\mathtt{p} = \mathtt{p}_0$ and $\mathtt{p} = \mathtt{p}_1$ are proved in the same way, except less rewriting of max expressions are required.

Sync: $G = \{l : G_l\}_{l\in L;L'}$

There are two subcases depending on $\mathtt{p}$.

If $\mathtt{p} \neq n$ then

$$\mathscr{G}^\star\langle n, m\rangle(\{l : G_l\}_{l\in L;L'})\upharpoonright\mathtt{p}$$

$$= (\mathtt{1} \to \mathtt{n{+}1} : m+1\{\text{cases}_{L_1'' \cup L'} \ldots \mathtt{n} \to \mathtt{n{+}1} : m+n\{\text{cases}_{L_n'' \cup L'} : \mathtt{n{+}1} \to \mathtt{1} : m+1\{l : \mathtt{1} \to \mathtt{2} : m+2 : \{l : \ldots \mathtt{n{-}1} \to \mathtt{n} : m+n\{l : \mathscr{G}^\star\langle n,m\rangle(G_l)\}\ldots\} \}_{l \in L' \cup (L_1'' \cap \ldots L_n'')}\}_{L_n'' \subseteq L} \ldots\}_{L_1'' \subseteq L})\restriction \mathtt{p}$$

$$\text{(rewriting max)} = \max_{\leq_{\max}}\{T \mid T \leq_{sub} m+\mathtt{p} \oplus \{\text{cases}_{L_\mathtt{p}'' \cup L'} : m+\mathtt{p}\&\{l : m+\mathtt{p}+1 \oplus \{l : \mathscr{G}^\star\langle n,m\rangle(G_l)\restriction \mathtt{p}\}\}_{l \in L' \cup (L_1'' \cap \ldots L_n'')}\}_{L_\mathtt{p}'' \subseteq L} \quad \forall L_1'', \ldots, L_{\mathtt{p}-1}'', L_{\mathtt{p}+1}'', \ldots L_n'' \subseteq L\}$$

$$= m+\mathtt{p} \oplus \{\text{cases}_{L_\mathtt{p}'' \cup L'} : m+\mathtt{p}\&\{l : m+\mathtt{p}+1 \oplus \{l : \mathscr{G}^\star\langle n,m\rangle(G_l)\restriction \mathtt{p}\}\}_{l \in L' \cup L_\mathtt{p}''}\}_{L_\mathtt{p}'' \subseteq L}$$

Now this case follows by the induction hypothesis.

The case where $\mathtt{p} = n$ is proves in the same way.

We have now proved the interesting cases therefore the lemma is fulfilled. □

A.4 Proof of Theorem 4.4[Type Preservation]

We Prove

$$\text{If } \frac{\mathscr{D}}{\Gamma \vdash P \rhd \Delta} \text{ then } \mathscr{G}_{\text{map}}(\Gamma) \vdash \mathscr{E}(\mathscr{D}) \rhd \mathscr{T}_{\text{map}}(\Delta)$$

By induction on the type derivation $\mathscr{D}$. The interesting cases Mcast and Sync are explained briefly.

Mcast:

$$\mathscr{D} = \frac{\Gamma \vdash a : \langle G\rangle \quad \dfrac{\mathscr{D}_1}{\Gamma \vdash P \rhd \Delta, \tilde{s} : (G\restriction \mathtt{1})@(\mathtt{1},n)} \quad |\tilde{s}| = \mathsf{max}(\mathsf{sid}(G)) \quad n = \mathsf{max}(\mathsf{pid}(G))}{\Gamma \vdash \overline{a}[2..n](\tilde{s}).P \rhd \Delta}$$

From this we obtain from induction hypothesis and $\mathscr{D}_1$ that

$$\frac{\mathscr{D}_1'}{\mathscr{G}_{\text{map}}(\Gamma) \vdash \mathscr{E}(\mathscr{D}_1) \rhd \mathscr{T}_{\text{map}}(\Delta), \tilde{s}, c_{\tilde{s}\mathtt{1}}, \ldots, c_{\tilde{s},\mathtt{n}} : \mathscr{T}\langle n, |\tilde{s}|, \mathtt{p}\rangle(G\restriction \mathtt{1})@(\mathtt{1},n+1)}$$

and therefore Lemma 4.3 gives us that

$$\frac{\mathscr{D}_1'}{\mathscr{G}_{\text{map}}(\Gamma) \vdash \mathscr{E}(\mathscr{D}_1) \rhd \mathscr{T}_{\text{map}}(\Delta), \tilde{s}, c_{\tilde{s}\mathtt{1}}, \ldots, c_{\tilde{s},\mathtt{n}} : \mathscr{G}^\star\langle n, |\tilde{s}|\rangle(G)\restriction \mathtt{1}@(\mathtt{1},n+1)}.$$

Now we can create

$$\mathscr{D}' = \text{Mcast}\frac{\mathscr{G}_{\text{map}}(\Gamma) \vdash a : \langle \mathscr{G}(G)\rangle \quad \mathscr{D}_1' \quad |\tilde{s}, c_{\tilde{s}\mathtt{1}}, \ldots, c_{\tilde{s}\mathtt{n}}| = n+m = \mathsf{max}(\mathsf{sid}(\mathscr{G}(G))) \quad n = \mathsf{max}(\mathsf{pid}(\mathscr{G}(G)))}{\mathscr{G}_{\text{map}}(\Gamma) \vdash \overline{a}[\mathtt{2..n{+}1}](\tilde{s}, c_{\tilde{s}\mathtt{1}}, \ldots, c_{\tilde{s}\mathtt{n}}).\mathscr{E}(\mathscr{D}_1) \rhd \mathscr{T}_{\text{map}}(\Delta)}.$$

Corollary 4.2 gives us that

$$\frac{\mathscr{D}_2'}{\mathscr{G}_{\text{map}}(\Gamma) \vdash \mathscr{C}\langle \tilde{s}, n\rangle(G) \rhd \emptyset}$$

and therefore we can prove the desired by

$$\text{Conc}\frac{\dfrac{\mathscr{D}_2'}{\mathscr{G}_{\text{map}}(\Gamma) \vdash \mathscr{C}\langle \tilde{s}, n\rangle(G) \rhd \emptyset} \quad \dfrac{\mathscr{D}'}{\mathscr{G}_{\text{map}}(\Gamma) \vdash \overline{a}[\mathtt{2..n{+}1}](\tilde{s}, c_{\tilde{s}\mathtt{1}}, \ldots, c_{\tilde{s}\mathtt{n}}).\mathscr{E}(\mathscr{D}_1) \rhd \mathscr{T}_{\text{map}}(\Delta)}}{\mathscr{G}_{\text{map}}(\Gamma) \vdash \mathscr{E}(\mathscr{D}) \rhd \mathscr{T}_{\text{map}}(\Delta)}$$

Sync:

$$\mathscr{D} = \text{Sync}\frac{\dfrac{\mathscr{D}_l}{\Gamma \vdash P_l \rhd \Delta, \tilde{s} : T_l@(\mathtt{p},n)} \quad \forall l \in L \quad L'' \subseteq L \cup L' \quad L' \subseteq L''}{\Gamma \vdash \{l : P_l\}_{l \in L''} \rhd \Delta, \tilde{s} : \{l : T_l\}_{l \in L;L'}@(\mathtt{p},n)}$$

There are two subcases depending on $\mathtt{p}$. If $\mathtt{p} = n$ then we get the type derivation by applying the Branch rule to the results of the induction hypothesis, and then applying a Sel rule to that. If $\mathtt{p} \neq n$ then we first apply the Sel rule to the results of the induction hypothesis, and then proceeds in the same way.

We have now proved the interesting cases therefore the theorem is fulfilled. □

A.5 Extension of Lemma 5.18 [Permutation] from [5]

(1) If $\dfrac{\dfrac{\dfrac{\mathscr{D}}{\Gamma \vdash P \rhd_{\tilde{t}} \Delta}\,\text{Subs}}{\Gamma \vdash P \rhd_{\tilde{t}} \Delta'}\,\text{Subs}}{\Gamma \vdash P \rhd_{\tilde{t}} \Delta''}$ then $\dfrac{\dfrac{\mathscr{D}}{\Gamma \vdash P \rhd_{\tilde{t}} \Delta}\,\text{Subs}}{\Gamma \vdash P \rhd_{\tilde{t}} \Delta''}$ and the result of $\mathscr{E}$ is unchanged.

(2) If $\dfrac{\dfrac{\dfrac{\mathscr{D}}{\Gamma \vdash P \rhd_{\tilde{t}} \Delta}\,\text{X}}{\Gamma \vdash P \rhd_{\tilde{t}} \Delta'}\,\text{Subs}}$ and the second last rule-application X is not Sel or Branch then the last two rule-applications can be permuted and the result of $\mathscr{E}$ is unchanged.

PROOF:
(1) Is immediate because $\leq_{\text{sub}}$ is transitive and $\mathscr{E}$ in both cases reduces to $\mathscr{E}(\mathscr{D})$.
(2) I proved for each possible rule X. This is done as in the original proof, where preservation of $\mathscr{E}$ is shown by evaluation since $\mathscr{E}\left(\dfrac{\dfrac{\mathscr{D}'}{\Gamma \vdash P \rhd_{\tilde{t}} \Delta'}\,\text{Subs}}\right) = \mathscr{E}(\mathscr{D}')$. There is one new case, and we will prove it now.
Sync: In this case we consider a derivation

$$\dfrac{\dfrac{\dfrac{\forall l \in L'' \quad \dfrac{\mathscr{D}_l}{\Gamma \vdash P_l \rhd_{\tilde{t}} T_l @ (\mathtt{p}, n)}}{\Gamma \vdash \mathtt{sync}_{\tilde{s},n}\{l : P_l\}_{l \in L''} \rhd_{\tilde{t}} \Delta, \tilde{s} : \{\{l : T_l\}_{l \in L; L'} @ (\mathtt{p}, n)\}}\,\text{Sync}}{\Gamma \vdash \mathtt{sync}_{\tilde{s},n}\{l : P_l\}_{l \in L''} \rhd_{\tilde{t}} \Delta', \tilde{s} : \{\{l : T'_l\}_{l \in L; L'} @ (\mathtt{p}, n)\}}\,\text{Subs}}$$

where $T_l \leq_{\text{sub}} T'_l$ for each $l \in L''$ and $\Delta \leq_{\text{sub}} \Delta'$. We can therefore create

$$\dfrac{\forall l \in L'' \quad \dfrac{\dfrac{\mathscr{D}_l}{\Gamma \vdash P_l \rhd_{\tilde{t}} T_l @ (\mathtt{p}, n)}\,\text{Subs}}{\Gamma \vdash \{l : P_l\}_{l \in L''} \rhd_{\tilde{t}} \Delta', \tilde{s} : \{T'_l @ (\mathtt{p}, n)\}}}{\Gamma \vdash \mathtt{sync}_{\tilde{s},n}\{l : P_l\}_{l \in L''} \rhd_{\tilde{t}} \Delta', \tilde{s} : \{\{l : T'_l\}_{l \in L; L'} @ (\mathtt{p}, n)\}}\,\text{Sync}$$

Now we only need to show that $\mathscr{E}$ is the same for both derivations, but this is fulfilled, since $\mathscr{E}(\dfrac{\dfrac{\mathscr{D}}{\Gamma \vdash P \rhd_{\tilde{t}} \Delta}\,\text{Subs}}) = \mathscr{E}(\mathscr{D})$. □

A.6 Extension of Theorem 5.22(1) from [5]

If $P \equiv Q$ and $\dfrac{\mathscr{D}_1}{\Gamma \vdash P \rhd_{\tilde{t}} \Delta}$ then there is a derivation $\dfrac{\mathscr{D}_2}{\Gamma \vdash Q \rhd_{\tilde{t}} \Delta}$ such that $\mathscr{E}(\mathscr{D}_1) \equiv \mathscr{E}(\mathscr{D}_2)$.

PROOF: This is proved by ruleinduction on $P \equiv Q$. All the rules are considered in [5], and we get that $\mathscr{E}(\mathscr{D}_1) \equiv \mathscr{E}(\mathscr{D}_2)$ for the derivations found in the original proof, by moving the conductor processes around and applying the considered equivalence rule. □

A.7 Proof of Theorem 4.5

This theorem is an extension of Theorem 5.22(2) from [5].
We prove

If $\dfrac{\mathscr{D}}{\Gamma \vdash P \rhd_{\tilde{t}} \Delta}$, $P \to P'$, Δ coherent, $\Delta \circ \Delta^\circ$ complete and $P_C \in \mathtt{PC}(\Delta \circ \Delta^\circ)$

then there is a derivation $\dfrac{\mathscr{D}'}{\Gamma \vdash P' \rhd_{\tilde{t}} \Delta'}$ and a $P'_C \in \mathtt{PC}(\Delta' \circ \Delta^\circ)$ where $\Delta = \Delta'$ or $\Delta \to \Delta'$

such that $\mathscr{E}(\mathscr{D})|P_C \to^\star \mathscr{E}(\mathscr{D}')|P'_C$.

By induction on the derivation of $P \to P'$. All cases except Sync are covered by the original proof, where $\mathscr{E}(\mathscr{D})|P_C \to^\star \mathscr{E}(\mathscr{D}')|P'_C$ can be proved for the found derivation $\mathscr{D}'$ by selecting P'_C using the same global types as was used to find P_C. We will show the case for Link as an example, and prove the new case for Sync.

Link: $\text{Link}\dfrac{}{\vdash \overline{a}[2..\mathtt{n}](\tilde{s}).P_1|a[2](\tilde{s}).P_2|\ldots|a[n](\tilde{s}).P_n \to (\nu\tilde{s})(P_1|P_2|\ldots|P_n|s_1:\emptyset|\ldots|s_m:\emptyset)}$

We can assume that the typing derivation $\mathscr{D}$ starts with n applications of the Conc rule without changing $\mathscr{E}(\mathscr{D})$ because of the extension of Lemma 5.18. The first Conc rule contains a derivation

$$\text{Mcast}\frac{\Gamma \vdash a : \langle G \rangle \quad \dfrac{\mathscr{D}_1}{\Gamma \vdash P_1 \rhd \Delta, \tilde{s} : (G\upharpoonright 1)@(1,n)} \quad |\tilde{s}| = \mathsf{max}(\mathsf{sid}(G)) \quad n = \mathsf{max}(\mathsf{pid}(G))}{\Gamma \vdash \overline{a}[2..\mathtt{n}](\tilde{s}).P_1 \rhd \Delta_1}$$

and the other Conc rules contain the derivations

$$\text{Macc}\frac{\Gamma \vdash a : \langle G \rangle \quad \dfrac{\mathscr{D}_\mathtt{p}}{\Gamma \vdash P \rhd \Delta_\mathtt{p}, \tilde{s} : (G\upharpoonright \mathtt{p})@(\mathtt{p},n)} \quad |\tilde{s}| = \mathsf{max}(\mathsf{sid}(G)) \quad n = \mathsf{max}(\mathsf{pid}(G))}{\Gamma \vdash a[\mathtt{p}](\tilde{s}).P_\mathtt{p} \rhd \Delta_\mathtt{p}} \text{ for } \mathtt{p} = 1..n.$$

We can now create $\mathscr{D}'$ as an application of the CRes rule containing $2 \cdot n$ applications of the Conc rule containing $\mathscr{D}_i$ for $i = 1..n$, and $\mathscr{D}_{n+1} \ldots \mathscr{D}_{2 \cdot n}$ which are derivations for the empty queues. $\mathscr{D}'$ concludes that

$$\Gamma \vdash (\nu\tilde{s})(P_1|P_2|\ldots|P_n|s_1:\emptyset|\ldots|s_n:\emptyset) \rhd \Delta.$$

Now can now use the definition of $\mathscr{E}$ to find

$$\mathscr{E}(\mathscr{D}) = a[\mathtt{n+1}](\tilde{s}, c_{\tilde{s}1}, \ldots, c_{\tilde{s}\mathtt{n}}).\mathscr{C}^\star\langle \tilde{s}, n\rangle|\overline{a}[2..\mathtt{n+1}](\tilde{s}).\mathscr{E}(\mathscr{D}_1)|a[2](\tilde{s}).\mathscr{E}(\mathscr{D}_2)|\ldots|a[\mathtt{n}](\tilde{s}).\mathscr{E}(\mathscr{D}_n)$$
$$\mathscr{E}(\mathscr{D}') = (\nu\tilde{s}, c_{\tilde{s}1}, \ldots, c_{\tilde{s}\mathtt{n}})(\mathscr{C}(G) \mid c_{\tilde{s}1} : \emptyset \mid \ldots \mid c_{\tilde{s}\mathtt{n}} : \emptyset \mid \mathscr{E}(\mathscr{D}_1)|\ldots|\mathscr{E}(\mathscr{D}_n)|s_1:\emptyset|\ldots|s_n:\emptyset)$$

With these choices of $\mathscr{D}$, $\mathscr{D}'$ the theorem is proved by a single Link step wrapped in a Conc and Str rule since we can choose $P_C = P'_C$.

Sync: $\text{Sync}\dfrac{l' \in \bigcap_{i=1}^n L_i}{\mathtt{sync}_{\tilde{s},n}\{l : P_{1l}\}_{l \in L_1} \mid \ldots \mid \mathtt{sync}_{\tilde{s},n}\{l : P_{nl}\}_{l \in L_n} \to P_{1l'} \mid \ldots \mid P_{nl'}}$

We can assume that the typing derivation $\mathscr{D}$ starts with n applications of the Conc rule without changing $\mathscr{E}(\mathscr{D})$ because of the extension of Lemma 5.18. The Conc rules contain the derivations

$$\text{Sync}\frac{\forall l \in L'' : \dfrac{\mathscr{D}_{\mathtt{p}l}}{\Gamma \vdash P_{\mathtt{p}l} \rhd \Delta_\mathtt{p}, \tilde{s} : \{T_{\mathtt{p}l}@(\mathtt{p},n)\}} \quad L'' \subseteq L \cup L' \quad L' \subseteq L''}{\Gamma \vdash \mathtt{sync}_{\tilde{s},n}\{l : P_{\mathtt{p}l}\}_{l \in L''} \rhd \Delta_\mathtt{p}, \tilde{s} : \{\{l : T_{\mathtt{p}l}\}_{l \in L;L'}@(\mathtt{p},n)\}} \text{ for } \mathtt{p} = 1..n.$$

We can now create $\mathscr{D}'$ as n applications of the Conc rule containing $\mathscr{D}_{1l'}, \ldots, \mathscr{D}_{nl'}$. $\mathscr{D}'$ concludes that $\Gamma \vdash P_{1l'} \mid \ldots \mid P_{nl'} \rhd \Delta_1, \tilde{s} : \{T_{\mathtt{p}l'}@(\mathtt{p},n)\}_{\mathtt{p} \in \{1..n\}}$.

Let $P_C \in \mathtt{PC}(\Delta) = \mathtt{PC}(\Delta_1, \tilde{s} : \{\{l : T_{\mathtt{p}l}\}_{l \in L;L'}@(\mathtt{p},n)\}_{\mathtt{p} \in \{1..n\}})$
$= \bigcup_{P_{C1} \in \mathtt{PC}(\Delta_1)} \{\mathscr{C}^\star\langle \tilde{s}, n\rangle(\{l : G_l\}_{l \in L;L'}|P_{C1}|c_{\tilde{s}1} : \emptyset|\ldots|c_{\tilde{s}\mathtt{n}} : \emptyset \mid G_l\upharpoonright \mathtt{p} = T_{\mathtt{p}l}\ \forall \mathtt{p} \in \{1..n\}, l \in L \cup L'\}$.

We can now choose $P'_C = \mathscr{C}^\star\langle \tilde{s}, n\rangle(G_l)|P_{C1}|c_{\tilde{s}1} : \emptyset|\ldots|c_{\tilde{s}\mathtt{n}} : \emptyset$.

With these choices of $\mathscr{D}$, $\mathscr{D}'$, P_C and P'_C the theorem is proved by performing the communication which $\mathscr{E}$ and $\mathtt{PC}$ produces from the synchronisation constructor. □

A.8 Proof of Corollary 4.6 [Semantics Preservation]

This corollary is an extension of Theorem 5.22(3) from [5].

We prove

If $\dfrac{\mathscr{D}}{\Gamma \vdash P \triangleright \emptyset}$ and $P \to^\star P'$

then there is a derivation $\dfrac{\mathscr{D}'}{\Gamma \vdash P' \triangleright \emptyset}$

such that $\mathscr{E}(\mathscr{D}) \to^\star \mathscr{E}(\mathscr{D}')$.

By induction on the number of steps in $P \to^\star P'$.

If $P = P'$ then the theorem is trivially fulfilled.

If $P \to^\star P'$ is of the form $P \to P_1 \to^\star P'$ then the extension of Theorem 5.22(2) gives us that there is a derivation $\dfrac{\mathscr{D}_1}{\Gamma \vdash P_1 \triangleright \emptyset}$ such that $\mathscr{E}(\mathscr{D})|0 \to^\star \mathscr{E}(\mathscr{D}_1)|0$, since $\emptyset$ is coherent and complete and $\mathtt{PC}(\emptyset) = \{0\}$. Now we can wrap each step in $\mathscr{E}(\mathscr{D})|0 \to^\star \mathscr{E}(\mathscr{D}_1)|0$ with a Str rule to get $\mathscr{E}(\mathscr{D}) \to^\star \mathscr{E}(\mathscr{D}_1)$.

The induction hypothesis yields that there is a derivation $\dfrac{\mathscr{D}'}{\Gamma \vdash P' \triangleright \emptyset}$ such that $\mathscr{E}(\mathscr{D}_1) \to^\star \mathscr{E}(\mathscr{D}')$, and therefore we get that $\mathscr{E}(\mathscr{D}) \to^\star \mathscr{E}(\mathscr{D}')$ by combining the steps in the two evaluations.

Therefore the theorem is fulfilled. □