Logical Methods in Computer Science
Vol. 13(2:12)2017, pp. 1–61
www.lmcs-online.org

Submitted Aug. 1, 2015
Published Jun. 30, 2017

ON THE PRECISENESS OF SUBTYPING IN SESSION TYPES *

TZU-CHUN CHEN [a], MARIANGIOLA DEZANI-CIANCAGLINI [a], ALCESTE SCALAS [c], AND NOBUKO YOSHIDA [d]

[a] Dept. of Computer Science, TU Darmstadt, Hochschulstr. 10, 64289 Darmstadt, Germany
e-mail address: tzu-chun.chen@dsp.tu-darmstadt.de

[b] Dip. di Informatica, Università di Torino, Corso Svizzera 185, 10149 Torino, Italy
e-mail address: dezani@di.unito.it

[c,d] Dept. of Computing, Imperial College London, 180 Queen's Gate, London SW7 2AZ, UK
e-mail address: {alceste.scalas, n.yoshida}@imperial.ac.uk

Abstract. Subtyping in concurrency has been extensively studied since early 1990s as one of the most interesting issues in type theory. The correctness of subtyping relations has been usually provided as the soundness for type safety. The converse direction, the completeness, has been largely ignored in spite of its usefulness to define the largest subtyping relation ensuring type safety. This paper formalises preciseness (i.e. both soundness and completeness) of subtyping for mobile processes and studies it for the synchronous and the asynchronous session calculi. We first prove that the well-known session subtyping, the branching-selection subtyping, is sound and complete for the synchronous calculus. Next we show that in the asynchronous calculus, this subtyping is incomplete for type-safety: that is, there exist session types T and S such that T can safely be considered as a subtype of S, but $T \leqslant S$ is not derivable by the subtyping. We then propose an asynchronous subtyping system which is sound and complete for the asynchronous calculus. The method gives a general guidance to design rigorous channel-based subtypings respecting desired safety properties. Both the synchronous and the asynchronous calculus are first considered with linear channels only, and then they are extended with session initialisations and communications of expressions (including shared channels).

Key words and phrases: Session types, Subtyping, Completeness, Soundness, the π-calculus, Type safety, Asynchronous message permutations.

* This work was partly supported by the COST Action IC1201 *BETTY*. .

[a] Tzu-chun Chen was supported by the ERC grant FP7-617805 *LiVeSoft*.

[d] Mariangiola Dezani was partly supported by EU projects H2020-644235 *Rephrase* and H2020-644298 *HyVar*, ICT COST Actions IC1402 , IC1405 and Ateneo/CSP project *RunVar*.

[c] Alceste Scalas was partly supported by EPSRC EP/K011715/1, and was also affiliated with: Dip. di Matematica e Informatica, Università di Cagliari, Italy.

[d] Nobuko Yoshida was partly supported by EPSRC EP/K011715/1, EP/K034413/1, EP/L00058X/1 and EP/N027833/1 and and EU FP7-612985 *UpScale*.

LOGICAL METHODS IN COMPUTER SCIENCE
DOI:10.23638/LMCS-13(2:12)2017

1. Introduction

Subtyping in concurrency. Since Milner first introduced the idea of assigning types to channels in the π-calculus [46], the subtypings which define an ordering over usages of channels have been recognised as one of the most useful concepts in the studies of the π-calculus.

The earliest work is a simple subtyping between input and output capabilities (called IO-subtyping) [57], which has been extended to and implemented in different areas of concurrency [30, 58, 59, 61] and has been continuously studied as one of the core subjects in concurrency [33, 34]. Later, a generic type system with subtyping was introduced by Igarashi *et al.* [41], where the subtyping plays a fundamental rôle in generating a variety of interesting type systems as its instances.

More recently, another subtyping based on session types [24] has been applied to many aspects of software design and implementations such as web services, programming languages and distributed computing [7, 15, 27, 31, 35, 39, 40, 60, 63]. The standpoint of session types is that communication-centred applications exhibit highly structured interactions involving, for example, sequencing, branching, selection and recursion, and such a series of interactions can be abstracted as a *session type* through a simple syntax. The session subtyping specified along session structures is then used for validating a large set of programs, giving flexibility to programmers.

As an example of session subtyping [7, 15, 39, 43], consider a simple protocol between a Buyer and a Seller from Buyer's viewpoint: Buyer sends a book's title (a string), and Seller sends a quote (an integer). If Buyer is satisfied by the quote, he then sends his address (a string) and Seller sends back the delivery date (a date); otherwise he quits the conversation. This can be described by the session type:

$$!\langle\mathtt{string}\rangle.?(\mathtt{int}).\{!\mathsf{ok}\langle\mathtt{string}\rangle.?(\mathtt{date}).\mathsf{end} \oplus !\mathsf{quit}.\mathsf{end}\} \tag{1.1}$$

The prefix $!\langle\mathtt{string}\rangle$ denotes an output of a value of type `string`, whereas $?(\mathtt{int})$ denotes an input of a value of type `int`. Instead ok and quit are labels distinguishing different branches. The operator $\oplus$ is an internal choice, meaning the process may choose to either send the label ok with a string and receive a date, or send the label quit. The type end represents the termination of the session. From Seller's viewpoint the same session is described by the *dual* type

$$?(\mathtt{string}).!\langle\mathtt{int}\rangle.\{?\mathsf{ok}(\mathtt{string}).!\langle\mathtt{date}\rangle.\mathsf{end} \,\&\, ?\mathsf{quit}.\mathsf{end}\} \tag{1.2}$$

in which & means that the process offers two behaviours: one where it receives ok with a string and sends a date, and one where it receives quit.

As $\mathtt{nat} \leqslant \mathtt{real}$ in the standard subtyping, a type representing a more confined behaviour is smaller. A selection subtype is a type which selects among fewer options (as outputs). The following is an example of a subtype of (1.1):

$$!\langle\mathtt{string}\rangle.?(\mathtt{int}).!\mathsf{ok}\langle\mathtt{string}\rangle.?(\mathtt{date}).\mathsf{end} \tag{1.3}$$

Conversely, a branching subtype is a type which offers more options (as inputs). The following is an example of a subtype of (1.2):

$$?(\mathtt{string}).!\langle\mathtt{int}\rangle.\{?\mathsf{ok}(\mathtt{string}).!\langle\mathtt{date}\rangle.\mathsf{end} \,\&\, ?\mathsf{quit}.\mathsf{end} \,\&\, ?\mathsf{later}.\mathsf{end}\} \tag{1.4}$$

Intuitively, a type T is a subtype of a type S if T is ready to receive no fewer labels than S, and T potentially sends no more labels than S (in other words, T represents a more

permissive behaviour than S) [7, 15]. If we run two processes typed by (1.3) and (1.4), they are *type safe*, i.e. there is no mismatch of labels or types during communication. Hence the subtyping is *sound* with respect to the type safety. An important question, however, is still remaining: is this subtyping *complete*? I.e. is this session subtyping the *largest* relation which does not violate type safety? The proof of soundness is usually immediate as a corollary of the subject reduction theorem. But how can we state and prove completeness?

A type system with a subsumption rule is parametric in the subtyping relation. A larger subtyping relation will yield a type system accepting more programs. A subtyping relation is *sound* if no typeable program is incorrect. It is *complete* if there is no strictly larger sound subtyping relation. Following Ligatti *et al.* [44], we say that a subtyping relation is *precise* if it is both sound and complete. The preciseness is a simple operational property that specifies a relationship between static and dynamic semantics.

Preciseness. To formally define preciseness, we assume a multi-step reduction between processes $P \to^* P'$ (where P' is possibly the `error` process) as well as typing judgements of the form $P \triangleright \{a : T\}$, assuring that the process P has a single free channel a whose type is T. We also use reduction contexts C in the standard way. The judgement $C[a : T] \triangleright \emptyset$ means that filling the hole of C with any process P typed by $a : T$ produces a well-typed closed process (formally $C[a : T] \triangleright \emptyset \iff X : T \vdash C[X\langle a\rangle] \triangleright \emptyset$, where X is a process variable which does not appear in C, see § 2).

Our preciseness definition is an adaptation of the preciseness definition for the call-by-value λ-calculus with sums and product types [44].

Definition 1.1 (Preciseness)**.** A subtyping $\leqslant$ is *precise* when, for all session types T, S:

$$T \leqslant S \iff \begin{pmatrix}\text{there do not exist } C \text{ and } P \text{ such that:}\\ C[a : S] \triangleright \emptyset \text{ and } P \triangleright \{a : T\} \text{ and } C[P] \to^* \texttt{error}\end{pmatrix}$$

When the *only-if* direction ($\Rightarrow$) of this formula holds, we say that the subtyping is *sound*; when the *if* direction ($\Leftarrow$) holds, we say that the subtyping is *complete*.

Consider the set of contexts C such that $C[a : S] \triangleright \emptyset$, i.e., contexts with one hole, where the channel a is typed as S. The *soundness* property of Definition 1.1 says that if we take *any* such C and fill it with *any* process with a typed as $T \leqslant S$, the result is safe. The *completeness* property, instead, says that for all $T \not\leqslant S$, we can find some C in the set above, and fill it with some process with a typed as T, so that their combination reduces to `error`. Soundness is clearly Liskov's Substitution Principle [45], whereas completeness ensures that a subtyping relation cannot be safely extended. Notice that we take a "must view" of correctness, asking that a correct process never reduces to an `error`.

Here we are interested in *syntactically* defined subtyping and an *operational* notion of preciseness. Our approach is opposed to semantic subtyping as proposed by Frisch *et al.* [21], which is given denotationally; in addition, the calculus introduced by Frisch *et al.* [21] has a type case constructor from which completeness follows for free. See § 8 for a detailed discussion.

Preciseness and impreciseness for the π-calculus. IO-subtyping classifies channels according to their reading and writing capabilities [57]. It is not precise, because no operational error can be detected when a read only channel is used to write, or vice versa, by a context without type annotations. For a similar reason, the branching and selection subtyping [15] is also imprecise for the π-calculus. The branching and selection subtyping

is instead precise for the π-calculus with only linear channels [42], whose expressivity is limited.

Igarashi *et al.* [41] only state necessary conditions for subtyping, the aim being that of having the maximum generality. The subtyping relations in the instances of the generic type system depend on the properties (arity-mismatch check, race detection, static garbage-channel collection, deadlock detection) one wants to guarantee.

These results led us to consider preciseness for two representative session calculi: the synchronous [36, 62] and the asynchronous [48, 49, 50, 51] session calculi.

Two preciseness results. Session types have sufficiently rich structure to assure completeness, hence if $T \not\leqslant S$, then T and S can be distinguished by suitable contexts and processes.

The first result of this article is preciseness of the branching-selection subtyping (dubbed also *synchronous subtyping*) described above for the synchronous session calculus. Our motivation to study the first result is to gently introduce a proof method for preciseness and justify the correctness of the synchronous subtyping, which is widely used in session-based calculi, programming languages and implementations [7, 15, 40, 60].

The case of the asynchronous session calculus is more challenging. The original session typed calculi [36] are based on synchronous communication primitives, assumed to be compiled into asynchronous interactions using queues — i.e., synchronous communications are modelled by asynchronous ones. Later researchers [38] found that, assuming *ordered* asynchronous communications for binary interactions, one could directly express asynchronous non-blocking interactions. One can then assure not only the original synchronous safety, but also the asynchronous safety, i.e. deadlock-freedom (every input process will always receive a message) and orphan message-freedom (every message in a queue will always be received by an input process).

Our first observation is that the branching-selection subtyping $\leqslant$ is not large enough for the asynchronous calculus, i.e. there exist session types T and S such that T can safely be considered as a subtype of S, but $T \leqslant S$ is not derivable by the subtyping. The reason is natural: in the presence of queues, the processes typed by the following two non-dual types can run in parallel *without* reducing to `error`:

$$\begin{aligned} T_a &= !\langle\texttt{int}\rangle.!\langle\texttt{char}\rangle.?(\texttt{string}).?(\texttt{nat}).\texttt{end} \\ T_b &= !\langle\texttt{string}\rangle.!\langle\texttt{nat}\rangle.?(\texttt{int}).?(\texttt{char}).\texttt{end} \end{aligned}$$

since a process typed by T_a can put two messages typed by `int` and `char` in one queue and a process typed by T_b can put two messages typed by `string` and `nat` in another queue, and they can receive the two messages from each queue, without getting stuck. Moreover, if we replace T_a or T_b with any of their respective subtypes under $\leqslant$, we can still safely compose the corresponding processes in parallel, without reducing to `error`: therefore, if we extend $\leqslant$ to also relate T_a and the dual of T_b, we would not compromise type safety.

The *asynchronous subtyping* proposed by Mostrous *et al.* [48, 49, 50, 51] permutes the order of messages, for example:

$$T_a \leqslant ?(\texttt{string}).?(\texttt{nat}).!\langle\texttt{int}\rangle.!\langle\texttt{char}\rangle.\texttt{end}$$

so that a process typed by T_a can have a type which is dual of T_b by the subsumption rule. This asynchronous permutation is often used as a means of messaging optimisation, e.g. as "messaging overlapping", in the parallel programming community [52, §6]. Our

result demonstrates the preciseness of this subtyping, which was introduced for practical motivations.

We have found that the subtyping of Mostrous' PhD thesis [48, 50] is unsound if we require the absence of orphan messages. If we allow orphan messages and we only have deadlock errors, then such a subtyping is sound but not complete. All this is discussed in § 8. The other asynchronous subtypings introduced by Mostrous *et al.* [49, 51], whose targets are the higher-order π-calculus and the multiparty session types, respectively, are sound for deadlock and orphan message errors. Hence we simplify them and we adapt them to the binary session π-calculus.

Contributions. As far as we are aware, this is the first time that completeness of subtypings, which is solely based on (untyped) operational semantics, is formalised and proved in the context of mobile processes. We also demonstrate its applicability to two session type disciplines, the synchronous and the asynchronous ones. The most technical challenge is the proof of completeness for the asynchronous subtyping, which requires some ingenuity in the definition of its negation relation. Key in the proofs is the construction of processes which characterise types. These processes allow us to show also the denotational preciseness of both synchronous and asynchronous subtypings.

This article is an expanded version of a previous paper presented at PPDP 2014 [13], including detailed definitions and full proofs, which were omitted. In addition, we provide new results about the uniqueness of precise subtyping relations (Corollary 3.5, Theorem 5.10). Moreover, we include a new section (§ 6) dealing with session initialisation and with communication of expressions (including shared channels), which were not treated: this demonstrates that our approach smoothly generalises to the original calculus [36], showing that the invariance of shared channel types is precise.

Outline. § 2 defines the synchronous session calculus and its typing system, and proves soundness of the branching-selection subtyping $\leqslant_{\mathsf{s}}$. § 3 proposes a general scheme for showing completeness, and proves that $\leqslant_{\mathsf{s}}$ is the unique complete subtyping for the synchronous session calculus. § 4 defines the asynchronous session calculus and introduces a new asynchronous subtyping relation $\leqslant_{\mathsf{a}}$, which is shown to be sound. § 5 proves that $\leqslant_{\mathsf{a}}$ is the unique subtyping that both extends $\leqslant_{\mathsf{s}}$, and is complete for the asynchronous calculus. This last completeness proof is non-trivial, since the permutations introduced by the asynchronous subtyping rules make session types unstructured. § 6 extends both calculi with shared channels for session initialisation and communications of expressions as in paper [36]. The proof of operational preciseness gives us denotational preciseness of both synchronous and asynchronous subtypings, as shown in § 7. Related work and conclusion are the contents of § 8 and § 9, respectively.

2. Synchronous Session Calculus

This section starts by introducing syntax and semantics of a simplification of the most widely studied synchronous session calculus [36, 66]. Since our main focus is on subtypings between session types, we only allow exchanges of linear channels. The obtained calculus is similar to that presented by Vasconcelos in 2009 [62]. We then define the typing system and prove soundness of subtyping as defined in Definition 1.1. § 6 will consider session initialisations and communication of expressions (including shared channels).

$P ::=$	**Process**				$D ::=$	**Declaration**
$\mathbf{0}$	(nil)	$\mid$	$X\langle\tilde{u}\rangle$	(variable)	$X(\tilde{x}) = P$	
$\mid \sum_{i\in I} u?l_i(x_i).P_i$	(input)	$\mid$	$u!l\langle u'\rangle.P$	(output)		
$\mid P \mid P$	(parallel)	$\mid$	$P \oplus P$	(choice)	$u ::=$	**Identifiers**
$\mid$ `def` D `in` P	(definition)	$\mid$	$(\nu ab)P$	(restriction)	a	(linear channel)
$\mid$ `error`	(error)				$\mid x$	(channel variable)

Table 1: Syntax of synchronous processes.

2.1. **Syntax.** A *session* is a series of interactions between two parties, possibly with branching and recursion, and serves as a unit of abstraction for describing communication protocols. We use the following base sets: *channel variables*, ranged over by $x, y, z \ldots$; *linear channels*, ranged over by a, b; *identifiers* (channel variables and linear channels), ranged over by $u, u' \ldots$; *labels*, ranged over by $l, l', \ldots$; *process variables*, ranged over by $X, Y, \ldots$; and *processes*, ranged over by $P, Q \ldots$. The syntax is given in Table 1.

Session communications are performed between an output process $u!l\langle u'\rangle.P$ and an input process $\sum_{i\in I} u?l_i(x_i).P_i$ (the l_i are pairwise distinct), where the former sends a channel choosing one of the branches offered by the latter. In $\sum_{i\in I} u?l_i(x_i).P_i$ and $u!l\langle u'\rangle.P$ the identifier u is the *subject* of input and output, respectively. The choice $P \oplus Q$ internally chooses either P or Q. In many session calculi [7, 36, 51] the conditional plays the rôle of the choice. The process `def` D `in` P is a recursive agent and $X\langle\tilde{u}\rangle$ is a recursive variable. We postulate *guarded recursion*, i.e. recursive variables can occur in recursive agent declarations only after an input/output prefix; for instance, `def` $X(\tilde{x}) = Y\langle\tilde{u}\rangle$ `in` P and `def` $X(\tilde{x}) = Y\langle\tilde{u}\rangle \oplus P'$ `in` P are *not* syntactically valid, whereas `def` $X(\tilde{x}) = u!l\langle u'\rangle.Y\langle\tilde{u}''\rangle$ `in` P is valid. The process $(\nu ab)P$ is a restriction which binds two channels, a and b in P, making them *co-channels*, i.e. allowing them to communicate (see rule [R-COM-SYNC] in Table 3). This double-restriction is commonly used in the recent literature of session types [24, 62]. We often omit $\mathbf{0}$ from the tail of processes.

The *bindings* for variables are in inputs and declarations, those for channels are in restrictions, and those for process variables are in declarations. The derived notions of bound and free identifiers, alpha equivalence, and substitution are standard. We use Barendregt convention [2, §2.1.13] that no bound name can occur free or in two different bindings.

By $\mathsf{fpv}(P)/\mathsf{fc}(P)$ we denote the set of *free process variables/free channels* in P. By $\mathsf{sc}(P)$ we denote the set of *free subject channels* in P, defined by:

$$\begin{array}{rcl} \mathsf{sc}(u!l\langle u'\rangle.P) & = & \mathsf{fc}(u) \cup \mathsf{sc}(P) \\ \mathsf{sc}(\sum_{i\in I} u?l_i(x_i).P_i) & = & \mathsf{fc}(u) \cup \bigcup_{i\in I} \mathsf{sc}(P_i) \\ \mathsf{sc}(\texttt{def } D \texttt{ in } P) & = & \mathsf{sc}(P) \end{array} \qquad \text{where:} \quad \begin{array}{l} \mathsf{fc}(x) = \emptyset \\ \mathsf{fc}(a) = \{a\} \end{array}$$

and as expected in the other cases. Note that we need to use $\mathsf{fc}(u)$, since u can be either a channel or a variable.

2.2. **Operational semantics.** Table 3 gives the reduction relation between synchronous processes which do not contain free variables. It uses the following evaluation context:

$$C[\,] ::= [\,] \;\Big|\; C[\,] \mid P \;\Big|\; (\nu ab)C[\,] \;\Big|\; \texttt{def } D \texttt{ in } C[\,]$$

and the structural rules of Table 2.

[S-PAR 1] $\mathbf{0} \mid P \equiv P$ [S-PAR 2] $P \mid Q \equiv Q \mid P$ [S-PAR 3] $P \mid (Q \mid R) \equiv (P \mid Q) \mid R$ [S-CH 1] $P \oplus Q \equiv Q \oplus P$ [S-CH 2] $(P \oplus Q) \oplus R \equiv Q \oplus (P \oplus R)$

[S-RES 1] $(\nu ab)\mathbf{0} \equiv \mathbf{0}$ [S-RES 2] $(\nu ab)P \mid Q \equiv (\nu ab)(P \mid Q)$ [S-RES 3] $(\nu ab)(\nu cd)P \equiv (\nu cd)(\nu ab)P$ [S-DEF 1] $\texttt{def}\ D\ \texttt{in}\ \mathbf{0} \equiv \mathbf{0}$

[S-DEF 2] $\texttt{def}\ D\ \texttt{in}\ (\nu ab)P \equiv (\nu ab)(\texttt{def}\ D\ \texttt{in}\ P)$ [S-DEF 3] $(\texttt{def}\ D\ \texttt{in}\ P) \mid Q \equiv \texttt{def}\ D\ \texttt{in}\ (P \mid Q)$

[S-DEF 4] $\texttt{def}\ D\ \texttt{in}\ \texttt{def}\ D'\ \texttt{in}\ P \equiv \texttt{def}\ D'\ \texttt{in}\ \texttt{def}\ D\ \texttt{in}\ P$

Table 2: Structural congruence for synchronous processes.

[R-COM-SYNC]
$$\frac{k \in I}{(\nu ab)(a!l_k\langle c\rangle.P \mid \sum_{i\in I} b?l_i(x_i).Q_i) \to (\nu ab)(P \mid Q_k\{^c/_{x_k}\})}$$

[R-DEF]
$$\texttt{def}\ X(\tilde{x}) = P\ \texttt{in}\ (X\langle\tilde{a}\rangle \mid Q) \to \texttt{def}\ X(\tilde{x}) = P\ \texttt{in}\ (P\{^{\tilde{a}}/_{\tilde{x}}\} \mid Q)$$

[R-CHOICE] $P \oplus Q \to P$ [R-CONTEXT] $\dfrac{P \to P'}{C[P] \to C[P']}$ [R-STRUCT] $\dfrac{P \equiv P' \quad P' \to Q' \quad Q' \equiv Q}{P \to Q}$

Table 3: Reduction of synchronous processes.

[ERR-MISM-SYNC]
$$\frac{\forall i \in I : l \neq l_i}{(\nu ab)(a!l\langle c\rangle.P \mid \sum_{i\in I} b?l_i(x_i).Q_i) \to \texttt{error}}$$

[ERR-NEW-SYNC] $\dfrac{a \in \mathsf{sc}(P) \quad b \notin \mathsf{fc}(P)}{(\nu ab)P \to \texttt{error}}$ [ERR-CONTEXT] $C[\texttt{error}] \to \texttt{error}$

[ERR-OUT-OUT-SYNC] $(\nu ab)(a!l\langle c\rangle.P \mid b!l'\langle c'\rangle.Q) \to \texttt{error}$

[ERR-IN-IN-SYNC] $(\nu ab)(\sum_{i\in I} a?l_i(x_i).P_i \mid \sum_{j\in J} b?l'_j(x'_j).Q_j) \to \texttt{error}$

Table 4: Error reduction for synchronous processes.

In Table 3, [R-COM-SYNC] is the main communication rule between input and output at two co-channels a and b, where the label l_k is selected and channel c is instantiated into the k-th input branch. Other rules are standard.

We also define error reduction, which is crucial for stating the preciseness theorem. Our guideline in this definition (both for the synchronous and, later, for the asynchronous semantics) is the following sentence from a seminal paper by Honda *et al.* [36]: "The typeability

of a program ensures two possibly communicating processes always own compatible communication patterns." In our case it amounts to require the duality of the communications offered by co-channels.

The error reduction rules are listed in Table 4. Rule [ERR-MISM-SYNC] is a mismatch between the output and input labels. Rule [ERR-NEW-SYNC] represents an error situation where one of two co-channels (b) is missing. Rule [ERR-OUT-OUT-SYNC] gives an error when two co-channels are both subjects of outputs, destroying the duality of sessions. Similarly rule [ERR-IN-IN-SYNC] gives an error when two co-channels are both subjects of inputs. We do not consider errors due to non linear use of channels, since they are statically prevented by the typing rules and cannot be introduced by changing the definition of subtyping. Obviously, arbitrary processes can be stuck without reducing to `error`: this can happen, for instance, due to the lack of a companion process, or if a process interacts on multiple interleaved sessions. A simple example is the process $a?l(x)$, which is deadlocked; intriguing examples of deadlocks caused by session interleaving can be written using process variables with more than one parameter.

We denote by $\rightarrow_{\mathsf{s}}$ the reduction relation for the synchronous processes, generated by the rules in Tables 3 and 4, and by $\rightarrow_{\mathsf{s}}^*$ the reflexive and transitive closure of $\rightarrow_{\mathsf{s}}$.

Proposition 2.1, says that if a process can reduce to `error` in one step, then a different reduction produces a process with the same property.

Proposition 2.1. If $P \rightarrow_{\mathsf{s}} \texttt{error}$ and $P \rightarrow_{\mathsf{s}} P' \neq \texttt{error}$, then $P' \rightarrow_{\mathsf{s}} \texttt{error}$.

Proof. By cases on the rule giving $P \rightarrow_{\mathsf{s}} \texttt{error}$. The statement holds vacuously for rules [ERR-MISM-SYNC], [ERR-OUT-OUT-SYNC] and [ERR-IN-IN-SYNC] (Table 4): in such cases, $P \rightarrow_{\mathsf{s}} P'$ implies $P' = \texttt{error}$.

In the case [ERR-NEW-SYNC], we have $P = (\nu ab)P_0$ with $a \in \mathsf{sc}(P_0)$ and $b \notin \mathsf{fc}(P_0)$, and $P \rightarrow_{\mathsf{s}} P' \neq \texttt{error}$. Such a transition can only fire by rule [R-CONTEXT], and thus $P' = (\nu ab)P_1$ and $P_0 \rightarrow_{\mathsf{s}} P_1$. By induction on the derivation of the latter transition, we can verify that $a \in \mathsf{sc}(P_1)$ and $b \notin \mathsf{fc}(P_1)$. Hence, again by rule [ERR-NEW-SYNC], we conclude $(\nu ab)P_1 = P' \rightarrow_{\mathsf{s}} \texttt{error}$.

In the case [ERR-CONTEXT] we have $P = C[\texttt{error}]$. If $P \rightarrow_{\mathsf{s}} P' \neq \texttt{error}$, then the reduction is fired inside the context $C[\,]$: by induction on the derivation of the transition, we can verify that $\exists C'[\,] : P' = C'[\texttt{error}]$. Hence, again by rule [ERR-CONTEXT], we conclude $P' \rightarrow_{\mathsf{s}} \texttt{error}$. □

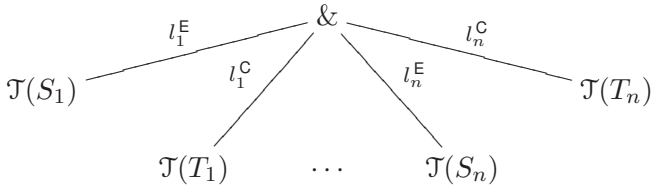
2.3. **Typing synchronous processes.** The syntax of *synchronous session types*, ranged over by T and S, is:

$$T, S \quad ::= \quad \&_{i\in I}?l_i(S_i).T_i \;\mid\; \bigoplus_{i\in I} !l_i\langle S_i\rangle.T_i \;\mid\; \mathbf{t} \;\mid\; \mu\mathbf{t}.T \;\mid\; \texttt{end}$$

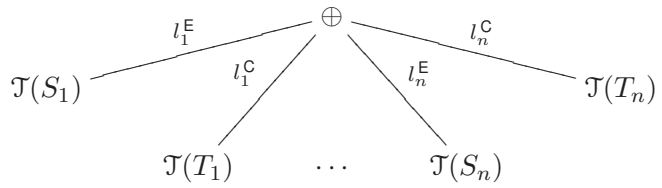
The *branching type* $\&_{i\in I}?l_i(S_i).T_i$ describes a channel willing to branch on an incoming label l_i, receive a channel of type S_i, and then continue its interaction as prescribed by T_i. The *selection type* $\bigoplus_{i\in I} !l_i\langle S_i\rangle.T_i$ is its dual: it describes a channel willing to send a label l_i with a channel of type S_i, and then continue its interaction as prescribed by T_i. In branching and in selection types:

- the labels are pairwise distinct;
- the types of the exchanged channels are closed.

$\mathcal{T}(\&_{1\leq i\leq n}?l_i(S_i).T_i) \;=$



$\mathcal{T}(\bigoplus_{1\leq i\leq n} !l_i\langle S_i\rangle.T_i) \;=$



$$\mathcal{T}(\mu\mathbf{t}.T) = \mathcal{T}(T\{^{\mu\mathbf{t}.T}/_{\mathbf{t}}\}) \qquad\qquad \mathcal{T}(\mathtt{end}) = \mathtt{end}$$

Table 5: Trees of session types.

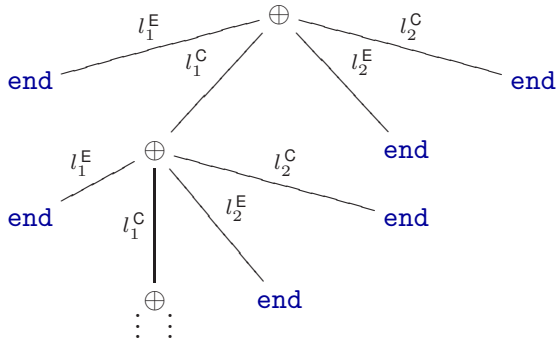


Figure 1: The tree of $\mu\mathbf{t}.!l_1\langle\mathtt{end}\rangle.\mathbf{t} \oplus !l_2\langle\mathtt{end}\rangle.\mathtt{end}$.

We omit & and $\oplus$ and labels when there is only one branch. We use $\mathbf{t}$ to range over type variables. The type $\mu\mathbf{t}.T$ is a *recursive type*. We assume that recursive types are *contractive*, i.e. $\mu\mathbf{t}_1.\mu\mathbf{t}_2 \ldots \mu\mathbf{t}_n.\mathbf{t}_1$ is not a type. The type $\mathtt{end}$ represents the termination of a session and it is often omitted.

We take an equi-recursive view of types [56, Chapter 20, §2], considering two types with the same regular tree as equal. Table 5 defines *coinductively* [56, Chapter 21, §2.1] the *tree of a type* T (notation $\mathcal{T}(T)$), where each label l generates two edges l^{E} and l^{C}, pointing respectively to the *exchanged* and the *continuation* sub-trees. Figure 1 shows (part of) the infinite tree $\mathcal{T}(\mu\mathbf{t}.!l_1\langle\mathtt{end}\rangle.\mathbf{t} \oplus !l_2\langle\mathtt{end}\rangle.\mathtt{end})$. We will mainly focus on *continuation paths*, i.e. (possibly infinite) sequences of edges $l_1^{\mathsf{C}}, \ldots, l_n^{\mathsf{C}}$ starting from tree roots.

In the examples we use infix notation for & and $\oplus$ and ground types ($\mathtt{int}, \mathtt{bool}, \ldots$) for messages. The extension to ground types is given in § 6. Unless otherwise noted, our definitions and statements will always refer to closed types.

As usual *session duality* [36] plays an important rôle for session types. The function $\overline{T}$, defined below, yields the dual of the (possibly open) session type T.

$$\overline{\&_{i\in I}?l_i(S_i).T_i} = \bigoplus_{i\in I} !l_i\langle S_i\rangle.\overline{T_i} \qquad \overline{\bigoplus_{i\in I} !l_i\langle S_i\rangle.T_i} = \&_{i\in I}?l_i(S_i).\overline{T_i}$$
$$\overline{\mathbf{t}} = \mathbf{t} \qquad \overline{\mu\mathbf{t}.T} = \mu\mathbf{t}.\overline{T} \qquad \overline{\mathtt{end}} = \mathtt{end}$$

We write $T_1 \bowtie T_2$ if $T_2 = \overline{T_1}$. Note that $\mathfrak{T}(\overline{T})$ can be obtained from $\mathfrak{T}(T)$ by turning branching nodes into selection nodes (and *vice versa*) in all continuation paths, without altering the exchanged sub-trees.

$$[\text{SUB-END}]\quad \mathtt{end} \leqslant \mathtt{end} \qquad [\text{SUB-BRA}]\quad \frac{\forall i \in I : S_i \leqslant S'_i \quad T_i \leqslant T'_i}{\&_{i\in I\cup J}?l_i(S_i).T_i \leqslant \&_{i\in I}?l_i(S'_i).T'_i} \qquad [\text{SUB-SEL}]\quad \frac{\forall i \in I : S'_i \leqslant S_i \quad T_i \leqslant T'_i}{\bigoplus_{i\in I} !l_i\langle S_i\rangle.T_i \leqslant \bigoplus_{i\in I\cup J} !l_i\langle S'_i\rangle.T'_i}$$

Table 6: Synchronous subtyping.

Table 6 defines the subtyping. Note that the double line in rules indicates that the rules should be interpreted *coinductively*. We follow the ordering of the branching-selection originally adopted by Honda *et al.* [7, 15, 39, 43, 49, 48, 50, 51]. Rule [SUB-BRA] states that the branching which offers fewer branches is a supertype of the one with more branches; and rule [SUB-SEL] is its dual (see the explanations in § 1). We write $T \leqslant_{\mathsf{s}} S$ if $T \leqslant S$ is derived by the rules in Table 6. Reflexivity of $\leqslant_{\mathsf{s}}$ is immediate and transitivity of $\leqslant_{\mathsf{s}}$ can be shown in the standard way, see Theorem A.1 in Appendix A.

The typing judgements for synchronous processes take the following form:

$$\Gamma \vdash_{\mathsf{s}} P \triangleright \Delta$$

where Γ is the *shared environment* which associates process variables to sequences of session types and Δ is the *session environment* which associates identifiers to session types. They are defined by:

$$\Gamma ::= \emptyset \mid \Gamma, X : \langle \tilde{T} \rangle \qquad \Delta ::= \emptyset \mid \Delta, u : T$$

We write Δ_1, Δ_2 for $\Delta_1 \cup \Delta_2$ when $\mathsf{dom}(\Delta_1) \cap \mathsf{dom}(\Delta_2) = \emptyset$. We say that Δ is *end-only* if $u : T \in \Delta$ implies $T = \mathtt{end}$.

We define a pre-order between the session environments which reflects subtyping. More precisely, $\Delta_1 \leqslant_{\mathsf{s}} \Delta_2$ if:

$$u \in \mathsf{dom}(\Delta_1) \cap \mathsf{dom}(\Delta_2) \text{ implies } \Delta_1(u) \leqslant_{\mathsf{s}} \Delta_2(u)$$
$$u \in \mathsf{dom}(\Delta_1) \text{ and } u \notin \mathsf{dom}(\Delta_2) \text{ imply } \Delta_1(u) = \mathtt{end}$$
$$u \notin \mathsf{dom}(\Delta_1) \text{ and } u \in \mathsf{dom}(\Delta_2) \text{ imply } \Delta_2(u) = \mathtt{end}$$

We write $\Delta_1 \approx_{\mathsf{s}} \Delta_2$ if $\Delta_1 \leqslant_{\mathsf{s}} \Delta_2$ and $\Delta_2 \leqslant_{\mathsf{s}} \Delta_1$. It is easy to verify that Δ is end-only iff $\Delta \approx_{\mathsf{s}} \emptyset$.

Table 7 gives the typing rules. They are standard in session calculi [24]. Rule [T-IDLE] is the introduction rule for the nil process. To type an input process, rule [T-INPUT] requires the type S_i of variable x_i and the type T_i of channel u for the continuation P_i. In the resulting session environment, the type u has the branching type in which u receives S_i and then continues with T_i for each label l_i. The rule for typing output processes is similar and simpler. In rule [T-PAR], the session environment of $P_1 \mid P_2$ is the disjoint union of

$$\frac{\Gamma \vdash P \triangleright \Delta \qquad \Delta \leqslant_{\mathsf{s}} \Delta'}{\Gamma \vdash P \triangleright \Delta'}\ [\text{T-SUB}] \qquad \Gamma \vdash \mathbf{0} \triangleright \emptyset\ [\text{T-IDLE}] \qquad \Gamma, X : \langle \tilde{T} \rangle \vdash X\langle \tilde{u} \rangle \triangleright \{\tilde{u} : \tilde{T}\}\ [\text{T-VAR}]$$

$$\frac{\forall i \in I : \Gamma \vdash P_i \triangleright \Delta, u : T_i, x_i : S_i}{\Gamma \vdash \sum_{i \in I} u?l_i(x_i).P_i \triangleright \Delta, u : \&_{i \in I} ?l_i(S_i).T_i}\ [\text{T-INPUT}] \qquad \frac{\Gamma \vdash P \triangleright \Delta, u : T}{\Gamma \vdash u!l\langle u' \rangle.P \triangleright \Delta, u : !l\langle S \rangle.T, u' : S}\ [\text{T-OUTPUT}]$$

$$\frac{\Gamma \vdash P_1 \triangleright \Delta_1 \qquad \Gamma \vdash P_2 \triangleright \Delta_2}{\Gamma \vdash P_1 \mid P_2 \triangleright \Delta_1, \Delta_2}\ [\text{T-PAR}] \qquad \frac{\Gamma \vdash P_1 \triangleright \Delta \qquad \Gamma \vdash P_2 \triangleright \Delta}{\Gamma \vdash P_1 \oplus P_2 \triangleright \Delta}\ [\text{T-CHOICE}]$$

$$\frac{\Gamma, X : \langle \tilde{T} \rangle \vdash P \triangleright \{\tilde{x} : \tilde{T}\} \qquad \Gamma, X : \langle \tilde{T} \rangle \vdash Q \triangleright \Delta}{\Gamma \vdash \mathtt{def}\ X(\tilde{x}) = P\ \mathtt{in}\ Q \triangleright \Delta}\ [\text{T-DEF}] \qquad \frac{\Gamma \vdash P \triangleright \Delta, a : T_1, b : T_2 \quad T_1 \bowtie T_2}{\Gamma \vdash (\nu ab)P \triangleright \Delta}\ [\text{T-NEW-SYNC}]$$

Table 7: Typing rules for synchronous processes.

the environments Δ_1 and Δ_2 for the two processes, reflecting the linear nature of channels. Contrarily, in rule [T-CHOICE], the two processes share the same session environment, since at most one of them will be executed. Rules [T-VAR] and [T-DEF] deal with process calls and definitions, requiring the channel parameters have the types which are assumed in the shared environment. Rule [T-VAR] gives these types to the arguments of the process variable. In [T-DEF], the parameters of the process associated with the process variable must be typed with these types. The assumption on parameter types is also used to type the body of the definition. Rule [T-NEW-SYNC] is a standard rule for name binding, where we ensure the co-channels have dual types. Finally, rule [T-SUB] is the standard rule for subtyping: a process P whose channels are typed according to the session environment Δ can be used in a type derivation requiring a less demanding environment Δ'. This is the key rule which allows, as described in § 1, to use a process P with a channel a of type (1.3) in a derivation where a has the supertype (1.1). As a consequence, a larger subtyping relation allows to type more processes.

We write $\Gamma \vdash_{\mathsf{s}} P \triangleright \Delta$ if P is typed using the rules in Table 7.

2.4. **Soundness of synchronous subtyping.** Our type system enjoys the standard property of subject reduction. Notice that session environments are unchanged since only bound channels can be reduced.

Theorem 2.2 (Subject reduction for synchronous processes)**.** *If* $\Gamma \vdash_{\mathsf{s}} P \triangleright \Delta$ *and* $P \to_{\mathsf{s}}^* Q$*, then* $\Gamma \vdash_{\mathsf{s}} Q \triangleright \Delta$*.*

From subject reduction we can easily derive that well-typed processes cannot produce error.

Corollary 2.3. *If* $\Gamma \vdash_{\mathsf{s}} P \triangleright \Delta$*, then* $P \not\to_{\mathsf{s}}^* \mathtt{error}$*.*

Proof. By Theorem 2.2, $P \to_{\mathsf{s}}^* \mathtt{error}$ implies $\Gamma \vdash_{\mathsf{s}} \mathtt{error} \triangleright \Delta$, which is impossible because `error` has no type. □

The proof of soundness theorem follows easily.

Theorem 2.4. *The synchronous subtyping relation $\leqslant_{\mathsf{s}}$ is sound for the synchronous calculus.*

The proofs of Theorems 2.2 and 2.4 are given in Appendix A.

3. Completeness for Synchronous Subtyping

This section proves the first main result, completeness of synchronous subtyping, which together with soundness shows the preciseness theorem. We shall take the following three steps.

- **[Step 1]** For each type T and identifier u, we define a *characteristic process* $\boldsymbol{P}(u, T)$ typed by $u : T$, which offers the series of interactions described by T on identifier u.
- **[Step 2]** We characterise the negation of the subtyping relation by inductive rules (notation $\not\leqslant_{\mathsf{s}}$).
- **[Step 3]** We leverage characteristic processes to prove that if $T \not\leqslant_{\mathsf{s}} S$, then there exist P, Q such that $P \triangleright \{a : T\}$ and $Q \triangleright \{b : \overline{S}\}$ and $(\nu ab)(P \mid Q) \rightarrow^* \texttt{error}$. Hence, by suitably choosing P, Q, and $C[\ ]$ in the definition of preciseness (Definition 1.1), we achieve completeness.

The same three steps will be used for the completeness proof in the asynchronous case.

Characteristic synchronous processes. The characteristic synchronous processes are defined following the structure of types. For each type we build a process with a single identifier offering the communications prescribed by the type. We also create auxiliary processes for exchanged identifiers.

Definition 3.1 (Characteristic synchronous processes)**.** The characteristic process offering communication T on identifier u for the synchronous calculus, denoted by $\boldsymbol{P}(u, T)$, is defined by induction on (possibly open) session types:

$$\boldsymbol{P}(u,T) \stackrel{\text{def}}{=} \begin{cases} \sum_{i\in I} \boldsymbol{P}_?(u, l_i, S_i, T_i) & \text{if } T = \&_{i\in I} ?l_i(S_i).T_i \\ \bigoplus_{i\in I} \boldsymbol{P}_!(u, l_i, S_i, T_i) & \text{if } T = \bigoplus_{i\in I} !l_i\langle S_i\rangle.T_i \\ X_{\mathbf{t}}\langle u\rangle & \text{if } T = \mathbf{t} \\ \texttt{def } X_{\mathbf{t}}(x) = \boldsymbol{P}(x, S) \texttt{ in } X_{\mathbf{t}}\langle u\rangle & \text{if } T = \mu\mathbf{t}.S \\ \mathbf{0} & \text{if } T = \texttt{end} \end{cases}$$

$$\boldsymbol{P}_?(u, l, S, T) \stackrel{\text{def}}{=} u?l(x).(\boldsymbol{P}(u, T) \mid \boldsymbol{P}(x, S))$$

$$\boldsymbol{P}_!(u, l, S, T) \stackrel{\text{def}}{=} (\nu ab)(u!l\langle a\rangle.\boldsymbol{P}(u, T) \mid \boldsymbol{P}(b, \overline{S}))$$

A branching type is mapped to the inputs $\boldsymbol{P}_?(u, l_i, S_i, T_i)$ $(i \in I)$, which uses the input channel x in $\boldsymbol{P}(x, S_i)$. A selection type is mapped to the choice between the outputs $\boldsymbol{P}_!(u, l_i, S_i, T_i)$ $(i \in I)$, where the fresh channel a carried by u will be received by the dual input, which will interact with the process $\boldsymbol{P}(b, \overline{S}_i)$. A recursive type is mapped in a definition associated to the characteristic process of the type body. The process body of this definition is just a call to the process variable associated to the recursive type variable. Type `end` is mapped to $\mathbf{0}$. Note that our characteristic processes interact sequentially on a *single* session — and in case of recursion, they only have one parameter: this is sufficient to

$$\frac{T \neq \mathtt{end}}{\mathtt{end} \not\leq T}\ [\text{N-END R}] \qquad \frac{T \neq \mathtt{end}}{T \not\leq \mathtt{end}}\ [\text{N-END L}]$$

$$[\text{N-BRASEL}]\ \&_{i\in I}?l_i(S_i).T_i \not\leq \bigoplus_{j\in J} !l'_j\langle S'_j\rangle.T'_j \qquad [\text{N-SELBRA-SYNC}]\ \bigoplus_{j\in J} !l'_j\langle S'_j\rangle.T'_j \not\leq \&_{i\in I}?l_i(S_i).T_i$$

$$\frac{\exists j \in J\ \forall i \in I : l_i \neq l'_j}{\&_{i\in I}?l_i(S_i).T_i \not\leq \&_{j\in J}?l'_j(S'_j).T'_j}\ [\text{N-LABEL-BRA}] \qquad \frac{\exists i \in I\ \forall j \in J : l_i \neq l'_j}{\bigoplus_{i\in I} !l_i\langle S_i\rangle.T_i \not\leq \bigoplus_{j\in J} !l'_j\langle S'_j\rangle.T'_j}\ [\text{N-LABEL-SEL}]$$

$$\frac{\exists i \in I\ \exists j \in J : l_i = l'_j\ S_i \not\leq S'_j}{\&_{i\in I}?l_i(S_i).T_i \not\leq \&_{j\in J}?l'_j(S'_j).T'_j}\ [\text{N-EXCH-BRA}] \qquad \frac{\exists i \in I\ \exists j \in J : l_i = l'_j\ S'_j \not\leq S_i}{\bigoplus_{i\in I} !l_i\langle S_i\rangle.T_i \not\leq \bigoplus_{j\in J} !l'_j\langle S'_j\rangle.T'_j}\ [\text{N-EXCH-SEL}]$$

$$\frac{\exists i \in I\ \exists j \in J : l_i = l'_j\ T_i \not\leq T'_j}{\&_{i\in I}?l_i(S_i).T_i \not\leq \&_{j\in J}?l'_j(S'_j).T'_j}\ [\text{N-CONT-BRA}] \qquad \frac{\exists i \in I\ \exists j \in J : l_i = l'_j\ T_i \not\leq T'_j}{\bigoplus_{i\in I} !l_i\langle S_i\rangle.T_i \not\leq \bigoplus_{j\in J} !l'_j\langle S'_j\rangle.T'_j}\ [\text{N-CONT-SEL}]$$

Table 8: Negation of synchronous subtyping.

capture the errors we are interested in, since they do not depend on multiple sessions being interleaved.

For example if $T = \mu\mathbf{t}.!l_1\langle\mathtt{end}\rangle.\mathbf{t} \oplus !l_2\langle !l_3\langle\mathtt{end}\rangle.\mathtt{end}\rangle.\mathtt{end}$, then

$$\boldsymbol{P}(a, T) \quad = \quad \mathtt{def}\ X_{\mathbf{t}}(x) = P\ \mathtt{in}\ X_{\mathbf{t}}\langle a\rangle$$

where

$$\begin{array}{rcl} P & = & \boldsymbol{P}_!(a, l_1, \mathtt{end}, \mathbf{t}) \oplus \boldsymbol{P}_!(a, l_2, !l_3\langle\mathtt{end}\rangle.\mathtt{end}, \mathtt{end}) \\ & = & (\nu bb')(a!l_1\langle b\rangle.\boldsymbol{P}(a, \mathbf{t}) \mid \boldsymbol{P}(b', \mathtt{end})) \oplus (\nu cc')(a!l_2\langle c\rangle.\boldsymbol{P}(a, \mathtt{end}) \mid \boldsymbol{P}(c', ?l_3(\mathtt{end}).\mathtt{end})) \\ & = & (\nu bb')(a!l_1\langle b\rangle.X_{\mathbf{t}}\langle a\rangle \mid \mathbf{0}) \oplus (\nu cc')(a!l_2\langle c\rangle.\mathbf{0} \mid c'?l_3(x).(\boldsymbol{P}(c', \mathtt{end}) \mid \boldsymbol{P}(x, \mathtt{end}))) \\ & = & (\nu bb')(a!l_1\langle b\rangle.X_{\mathbf{t}}\langle a\rangle \mid \mathbf{0}) \oplus (\nu cc')(a!l_2\langle c\rangle.\mathbf{0} \mid c'?l_3(x).(\mathbf{0} \mid \mathbf{0})) \\ & \equiv & (\nu bb')(a!l_1\langle b\rangle.X_{\mathbf{t}}\langle a\rangle) \oplus (\nu cc')(a!l_2\langle c\rangle \mid c'?l_3(x)) \end{array}$$

We can easily check that characteristic processes are well typed as expected.

Lemma 3.2. $\vdash_{\mathsf{s}} \boldsymbol{P}(u, T) \triangleright \{u : T\}$.

Rules for negation of synchronous subtyping. Table 8 defines the rules which characterise when a type is not a subtype of another type. We formulate these rules inductively, to simplify the completeness proof. Rules [N-END R] and [N-END L] say that `end` cannot be a super or subtype of a type different from `end`. Rule [N-BRASEL] says that a branching type cannot be subtype of a selection type. Rule [N-SELBRA-SYNC] is its dual. Rules [N-LABEL-BRA] and [N-LABEL-SEL] represent the cases in which the labels do not conform to the subtyping rules. Rules [N-EXCH-BRA] and [N-EXCH-SEL] represent the cases in which carried types do not match the subtyping rules. Lastly, rules [N-CONT-BRA] and [N-CONT-SEL] represent the cases in which continuations do not match the subtyping rules. Notice that if rule [SUB-BRA] holds, then the rules [N-⋆-BRA] do not hold, and if rule [SUB-SEL] holds, then the rules [N-⋆-SEL] do

not hold, where $\star \in \{\text{LABEL}, \text{EXCH}, \text{CONT}\}$. We write $T \not\leqslant_{\mathsf{s}} S$ if $T \not\leqslant S$ is derived by the rules in Table 8.

In Lemma 3.3, we show that $\not\leqslant_{\mathsf{s}}$ is the negation of the synchronous subtyping. This result will be used (in the *"only if"* direction) in the proof of Theorem 3.4.

Lemma 3.3. *If $T \leqslant_{\mathsf{s}} S$ is not derivable if and only if $T \not\leqslant_{\mathsf{s}} S$ is derivable.*

Proof. If $T \not\leqslant_{\mathsf{s}} S$, then we can show $T \not\leqslant_{\mathsf{s}} S$ by induction on the derivation of $T \not\leqslant_{\mathsf{s}} S$. We develop just two cases (the others are similar):

- base case [N-BRASEL]. Then, $T = \&_{i\in I}?l_i(S_i).T_i$ and $S = \bigoplus_{j\in J} !l'_j\langle S'_j\rangle.T'_j$. We can verify that T and S do not match the conclusion of [SUB-END], nor [SUB-BRA], nor [SUB-SEL] — hence, we conclude $T \not\leqslant_{\mathsf{s}} S$;
- inductive case [N-CONT-SEL]. Then, $T = \bigoplus_{i\in I} !l_i\langle S_i\rangle.T_i$ and $S = \bigoplus_{j\in J} !l'_j\langle S'_j\rangle.T'_j$; moreover, $\exists i \in I, j \in J : l_i = l'_j$ and $T_i \not\leqslant T'_j$ — and thus, by the induction hypothesis, $T_i \not\leqslant_{\mathsf{s}} T'_j$. We now notice that $T \leqslant_{\mathsf{s}} S$ could only possibly hold by rule [SUB-SEL] — but, since $T_i \not\leqslant_{\mathsf{s}} T'_j$, at least one of the coinductive premises of such a rule is not satisfied. Hence, we conclude $T \not\leqslant_{\mathsf{s}} S$.

Vice versa, if $T \not\leqslant_{\mathsf{s}} S$ we construct a derivation of $T \not\leqslant_{\mathsf{s}} S$ by looking at a "failing derivation" of $T \leqslant_{\mathsf{s}} S$. If we try to apply the subtyping rules to show $T \leqslant_{\mathsf{s}} S$, there exists a derivation branch that fails after n steps, i.e. that reaches two types T', S' whose trees do *not* match the conclusion of [SUB-END], nor [SUB-BRA], nor [SUB-SEL]. Note that no alternative derivation exists, because the rules in Table 6 do not overlap, hence at most one of them can be applied at each step. We prove $T \not\leqslant_{\mathsf{s}} S$ by induction on n, turning the failing coinductive derivation branch into a derivation of depth $n+1$ which concludes $T \not\leqslant_{\mathsf{s}} S$:

- base case $n = 0$. The derivation fails immediately, i.e. $T' = T$ and $S' = S$. By cases on the possible shapes of T and S, we construct a derivation which concludes $T \not\leqslant_{\mathsf{s}} S$ in $1 = n + 1$ steps, by one of the axioms [N-END R], [N-END L], [N-BRASEL], [N-SELBRA-SYNC], [N-LABEL-BRA], [N-LABEL-SEL];
- inductive case $n = m + 1$. The shapes of T, S match the conclusion of rule [SUB-BRA] (resp. [SUB-SEL]), but there is some coinductive premise $T' \leqslant_{\mathsf{s}} S'$ whose sub-derivation has a branch that fails after m steps. By the induction hypothesis, there exists a derivation of depth $m + 1$ that concludes $T' \not\leqslant_{\mathsf{s}} S'$; using this as a premise, by one of the rules [N-EXCH-BRA] or [N-CONT-BRA] (resp. [N-EXCH-SEL] or [N-CONT-SEL]), we construct a derivation of depth $(m + 1) + 1 = n + 1$ which concludes $T \not\leqslant_{\mathsf{s}} S$. □

The main theorem for synchronous subtyping can now be proved.

Theorem 3.4 (Completeness for synchronous subtyping)**.** *The synchronous subtyping relation $\leqslant_{\mathsf{s}}$ is complete for the synchronous calculus.*

Proof. We need to produce P, Q as described in **Step 3** on page 12. We let $P = \boldsymbol{P}(a, T)$ and $Q = \boldsymbol{P}(b, \overline{S})$, and we show that

$$(\nu ab)(\boldsymbol{P}(a, T) \mid \boldsymbol{P}(b, \overline{S})) \to^*_{\mathsf{s}} \texttt{error}$$

where $\boldsymbol{P}(a, T)$, $\boldsymbol{P}(b, \overline{S})$ are characteristic synchronous processes. The proof is by induction on the derivation of $T \not\leqslant_{\mathsf{s}} S$.

Case [N-END R]: $T = \texttt{end}$ and $S \neq \texttt{end}$.

$$(\nu ab)(\boldsymbol{P}(a, T) \mid \boldsymbol{P}(b, \overline{S})) = (\nu ab)(\mathbf{0} \mid \boldsymbol{P}(b, \overline{S})) \to_{\mathsf{s}} \texttt{error}$$

by rule [ERR-NEW-SYNC]: in fact, we have $a \notin \mathsf{fc}(\mathbf{0})$, and $S \neq \mathtt{end}$ implies that $\boldsymbol{P}(b, \overline{S})$ is a (possibly recursive) internal or external choice, and thus $b \in \mathsf{sc}(\boldsymbol{P}(b, \overline{S}))$ (by Definition 3.1).

Case [N-END L]: $T \neq \mathtt{end}$ and $S = \mathtt{end}$. The proof is as in the previous case.

Case [N-BRASEL]: $T = \&_{i\in I}?l_i(T'_i).T_i$ and $S = \bigoplus_{j\in J} !l'_j\langle S'_j\rangle.S_j$.

$$(\nu ab)(\boldsymbol{P}(a,T) \mid \boldsymbol{P}(b,\overline{S})) = (\nu ab)(\textstyle\sum_{i\in I} \boldsymbol{P}_?(a,l_i,T'_i,T_i) \mid \sum_{j\in J} \boldsymbol{P}_?(b,l'_j,S'_j,\overline{S_j})) \to_{\mathsf{s}} \mathtt{error}$$

by rule [ERR-IN-IN-SYNC].

Case [N-SELBRA-SYNC]: $T = \bigoplus_{i\in I} !l_i\langle T'_i\rangle.T_i$ and $S = \&_{j\in J}?l'_j(S'_j).S_j$.

$$(\nu ab)(\boldsymbol{P}(a,T) \mid \boldsymbol{P}(b,\overline{S})) = (\nu ab)(\textstyle\bigoplus_{i\in I} \boldsymbol{P}_!(a,l_i,T'_i,T_i) \mid \bigoplus_{j\in J} \boldsymbol{P}_!(b,l'_j,S'_j,\overline{S_j})) \to_{\mathsf{s}} \mathtt{error}$$

by rule [ERR-OUT-OUT-SYNC].

Case [N-LABEL-BRA]: $T = \&_{i\in I}?l_i(T'_i).T_i$, $S = \&_{j\in J}?l'_j(S'_j).S_j$, and $\exists k \in J$ such that $\forall i \in I : l'_k \neq l_i$.

$$(\nu ab)(\boldsymbol{P}(a,T) \mid \boldsymbol{P}(b,\overline{S})) = (\nu ab)(\textstyle\sum_{i\in I} \boldsymbol{P}_?(a,l_i,T'_i,T_i) \mid \bigoplus_{j\in J} \boldsymbol{P}_!(b,l'_j,S'_j,\overline{S_j})) \to_{\mathsf{s}}$$
$$(\nu ab)(\textstyle\sum_{i\in I} \boldsymbol{P}_?(a,l_i,T'_i,T_i) \mid \boldsymbol{P}_!(b,l'_k,S'_k,\overline{S_k})) \to_{\mathsf{s}} \mathtt{error}$$

by rule [ERR-MISM-SYNC].

Case [N-LABEL-SEL]: $T = \bigoplus_{i\in I} !l_i\langle T'_i\rangle.T_i$, $S = \bigoplus_{j\in J} !l'_j\langle S'_j\rangle.S_j$ and $\exists k \in I$ such that $\forall j \in J : l_k \neq l'_j$.

$$(\nu ab)(\boldsymbol{P}(a,T) \mid \boldsymbol{P}(b,\overline{S})) = (\nu ab)(\textstyle\bigoplus_{i\in I} \boldsymbol{P}_!(a,l_i,T'_i,T_i) \mid \sum_{j\in J} \boldsymbol{P}_?(b,l'_j,S'_j,\overline{S_j})) \to_{\mathsf{s}}$$
$$(\nu ab)(\boldsymbol{P}_!(a,l_k,T'_k,T_k) \mid \textstyle\sum_{j\in J} \boldsymbol{P}_?(b,l'_j,S'_j,\overline{S_j})) \to_{\mathsf{s}} \mathtt{error}$$

by rule [ERR-MISM-SYNC].

Case [N-EXCH-BRA]: $T = \&_{i\in I}?l_i(T'_i).T_i$, $S = \&_{j\in J}?l'_j(S'_j).S_j$, and $\exists k \in I\ \exists k' \in J$ such that $l_k = l'_{k'}$ and $T'_k \not\leftrightarrows_{\mathsf{s}} S'_{k'}$.

$$(\nu ab)(\boldsymbol{P}(a,T) \mid \boldsymbol{P}(b,\overline{S})) = (\nu ab)(\textstyle\sum_{i\in I} \boldsymbol{P}_?(a,l_i,T'_i,T_i) \mid \bigoplus_{j\in J} \boldsymbol{P}_!(b,l'_j,S'_j,\overline{S_j})) \to_{\mathsf{s}}$$
$$(\nu ab)(\textstyle\sum_{i\in I} \boldsymbol{P}_?(a,l_i,T'_i,T_i) \mid (\nu cd)(b!l_k\langle c\rangle.\boldsymbol{P}(b,\overline{S_{k'}}) \mid \boldsymbol{P}(d,\overline{S'_{k'}}))) \to_{\mathsf{s}}$$
$$(\nu ab)((\nu cd)(\boldsymbol{P}(a,T_k) \mid \boldsymbol{P}(c,T'_k) \mid \boldsymbol{P}(b,\overline{S_{k'}}) \mid \boldsymbol{P}(d,\overline{S'_{k'}}))) \equiv C[(\nu cd)(\boldsymbol{P}(c,T'_k) \mid \boldsymbol{P}(d,\overline{S'_{k'}}))]$$

where

$$C[\,] = (\nu ab)(\boldsymbol{P}(a,T_k) \mid \boldsymbol{P}(b,\overline{S_{k'}})) \mid [\,].$$

By induction,

$$(\nu cd)(\boldsymbol{P}(c,T'_k) \mid \boldsymbol{P}(d,\overline{S'_{k'}})) \to^*_{\mathsf{s}} \mathtt{error}$$

then by rule [ERR-CONTEXT], we conclude

$$C[(\nu cd)(\boldsymbol{P}(c,T'_k) \mid \boldsymbol{P}(d,\overline{S'_{k'}}))] \to^*_{\mathsf{s}} \mathtt{error}$$

Case [N-EXCH-SEL]: $T = \bigoplus_{i\in I} !l_i\langle T'_i\rangle.T_i$, $S = \bigoplus_{j\in J} !l'_j\langle S'_j\rangle.S_j$, and $\exists k \in I\ \exists k' \in J$ such that $l_k = l'_{k'}$ and $S'_{k'} \not\leftrightarrows_{\mathsf{s}} T'_k$.

$$(\nu ab)(\boldsymbol{P}(a,T) \mid \boldsymbol{P}(b,\overline{S})) \to_{\mathsf{s}} (\nu ab)(\nu cd)((a!l_k\langle c\rangle.\boldsymbol{P}(a,T_k) \mid \boldsymbol{P}(d,\overline{T'_k})) \mid \textstyle\sum_{j\in J} \boldsymbol{P}_?(b,l'_j,S'_j,\overline{S_j})) \to_{\mathsf{s}}$$
$$(\nu ab)(\nu cd)(\boldsymbol{P}(a,T_k) \mid \boldsymbol{P}(d,\overline{T'_k}) \mid \boldsymbol{P}(b,\overline{S_{k'}}) \mid \boldsymbol{P}(c,S'_{k'})) \equiv C[(\nu cd)(\boldsymbol{P}(c,S'_{k'}) \mid \boldsymbol{P}(d,\overline{T'_k}))]$$

where

$$C[\,] = (\nu ab)(\boldsymbol{P}(a,T_k) \mid \boldsymbol{P}(b,\overline{S_{k'}})) \mid [\,]$$

By induction,

$$(\nu cd)(\boldsymbol{P}(c,S'_{k'}) \mid \boldsymbol{P}(d,\overline{T'_k})) \to^*_{\mathsf{s}} \mathtt{error}$$

then by rule [ERR-CONTEXT], we conclude

$$C[(\nu cd)(\boldsymbol{P}(c, S'_{k'}) \mid \boldsymbol{P}(d, \overline{T'_k}))] \to^*_{\mathsf{s}} \texttt{error}$$

Case [N-CONT-BRA]: $T = \&_{i\in I}?l_i(T'_i).T_i$, $S = \&_{j\in J}?l'_j(S'_j).S_j$, and $\exists k \in I\ \exists k' \in J$ such that $l_k = l'_{k'}$ and $T_k \not\dashv_{\mathsf{s}} S_{k'}$. As in the previous case

$$(\nu ab)(\boldsymbol{P}(a, T) \mid \boldsymbol{P}(b, \overline{S})) \to^*_{\mathsf{s}} (\nu ab)(\nu cd)(\boldsymbol{P}(a, T_k) \mid \boldsymbol{P}(c, T'_k) \mid \boldsymbol{P}(b, \overline{S_{k'}}) \mid \boldsymbol{P}(d, \overline{S'_{k'}})) \equiv C[(\nu ab)(\boldsymbol{P}(a, T_k) \mid \boldsymbol{P}(b, \overline{S_{k'}}))]$$

where

$$C[\ ] = (\nu cd)(\boldsymbol{P}(c, T'_k) \mid \boldsymbol{P}(d, \overline{S'_{k'}})) \mid [\ ]$$

By induction,

$$(\nu ab)(\boldsymbol{P}(a, T_k) \mid \boldsymbol{P}(b, \overline{S_{k'}})) \to^*_{\mathsf{s}} \texttt{error}$$

then by rule [ERR-CONTEXT], we conclude

$$C[(\nu ab)(\boldsymbol{P}(a, T_k) \mid \boldsymbol{P}(b, \overline{S_{k'}}))] \to^*_{\mathsf{s}} \texttt{error}$$

Case [N-CONT-SEL]: $T = \bigoplus_{i\in I} !l_i\langle T'_i\rangle.T_i$, $S = \bigoplus_{j\in J} !l'_j\langle S'_j\rangle.S_j$, and $\exists k \in I\ \exists k' \in J$ such that $l_k = l'_{k'}$ and $T_k \not\dashv_{\mathsf{s}} S_{k'}$. As in the previous case

$$(\nu ab)(\boldsymbol{P}(a, T) \mid \boldsymbol{P}(b, \overline{S})) \to_{\mathsf{s}} (\nu ab)(\nu cd)(\boldsymbol{P}(a, T_k) \mid \boldsymbol{P}(b, \overline{S_{k'}}) \mid \boldsymbol{P}(c, S'_{k'}) \mid \boldsymbol{P}(d, \overline{T'_k})) \equiv C[(\nu ab)(\boldsymbol{P}(a, T_k) \mid \boldsymbol{P}(b, \overline{S_{k'}}))]$$

where

$$C = (\nu cd)(\boldsymbol{P}(c, S'_{k'}) \mid \boldsymbol{P}(d, \overline{T'_k})) \mid [\ ]$$

By induction,

$$(\nu ab)(\boldsymbol{P}(a, T_k) \mid \boldsymbol{P}(b, \overline{S_{k'}})) \to^*_{\mathsf{s}} \texttt{error}$$

then by rule [ERR-CONTEXT], we conclude

$$C[(\nu ab)(\boldsymbol{P}(a, T_k) \mid \boldsymbol{P}(b, \overline{S_{k'}}))] \to^*_{\mathsf{s}} \texttt{error}$$

Summing up, we proved that $T \not\leqslant_{\mathsf{s}} S$ implies $(\nu ab)(\boldsymbol{P}(a, T) \mid \boldsymbol{P}(b, \overline{S})) \to^*_{\mathsf{s}} \texttt{error}$: hence, the subtyping relation $\leqslant_{\mathsf{s}}$ is complete for the synchronous calculus, according to Definition 1.1. □

Corollary 3.5 shows that $\leqslant_{\mathsf{s}}$ is the unique precise subtyping for the synchronous calculus. Notably, this result is based on the definitions of typing system and preciseness we adopted for our treatment; a similar result could also be shown for the Gay-Hole type system and subtyping [24], as discussed in §8 (paragraph "Choices of typing system and subtyping").

Corollary 3.5. *$\leqslant_{\mathsf{s}}$ is the unique precise subtyping for the synchronous calculus.*

Proof. Take a reflexive and transitive relation $\sqsubseteq_{\mathsf{s}} \not\subseteq \leqslant_{\mathsf{s}}$ — i.e., $\exists S, T$ such that $S \sqsubseteq_{\mathsf{s}} T$ but $S \not\leqslant_{\mathsf{s}} T$. We prove that $\sqsubseteq_{\mathsf{s}}$ is an unsound subtyping. By the proof of Theorem 3.4, $T \not\leqslant_{\mathsf{s}} S$ implies $(\nu ab)(\boldsymbol{P}(a, T) \mid \boldsymbol{P}(b, \overline{S})) \to^*_{\mathsf{s}} \texttt{error}$. Therefore, by Definition 1.1, $\sqsubseteq_{\mathsf{s}}$ is not a sound subtyping. We conclude that if $\sqsubseteq_{\mathsf{s}}$ is sound, then $\sqsubseteq_{\mathsf{s}} \subseteq \leqslant_{\mathsf{s}}$; hence, $\leqslant_{\mathsf{s}}$ is the largest sound subtyping for the synchronous calculus, and therefore the unique precise one. □

4. Asynchronous Session Calculus

"Asynchrony" in communication means that message outputs are *non-blocking*. Given a pair of co-channels a and b, we model asynchrony using *two* FIFO queues: one queue delivers messages from a to b, and the other from b to a. This double-queue model *preserves message order*, resembling communication over a TCP/IP-like network: this is the most common formulation for asynchronous sessions, and follows recent formalisms in literature [14, 25, 39, 48, 49, 50].

4.1. **Syntax and operational semantics.** Table 9 shows the asynchronous session calculus obtained by extending the synchronous calculus of Table 1 with queues. A queue $ab \blacktriangleright h$ is used by channel a to enqueue messages in h and by channel b to dequeue messages from h. We extend the definition of the set of free channels to queues by $\mathsf{fc}(\varnothing) = \emptyset$, $\mathsf{fc}(ab \blacktriangleright h) = \{a, b\} \cup \mathsf{fc}(h)$, $\mathsf{fc}(l\langle a\rangle) = \{a\}$, and $\mathsf{fc}(h_1 \cdot h_1) = \mathsf{fc}(h_1) \cup \mathsf{fc}(h_2)$.

We use the structural congruence defined by adding the rules in Table 10 to the rules of Table 2. Rule [s-null] represents garbage collection of empty queues.

The reduction rules for asynchronous processes are obtained from the reduction rules of synchronous processes given in Table 3 by replacing rule [r-com-sync] with [r-send-async] and [r-receive-async], shown in Table 11. Rule [r-send-async] enqueues messages and rule [r-receive-async] dequeues messages.

We write $P \rightarrow_{\mathsf{a}} Q$ if $P \rightarrow Q$ is derived by the rules of Table 3 but rule [r-com-sync] and by the rules of Table 11.

$P ::=$		**Process**	h	$::=$		**Queue**			
	$\vdots$	from Table 1			$\varnothing$	(empty)	$\mid$	$l\langle a\rangle$	(message)
$\mid$	$ab \blacktriangleright h$	(queue)		$\mid$	$h \cdot h$	(composition)			

Table 9: Syntax of asynchronous processes.

$$[\text{s-null}] \quad (\nu ab)(ab \blacktriangleright \varnothing \mid ba \blacktriangleright \varnothing) \equiv \mathbf{0} \qquad [\text{s-queue-equiv}] \quad \frac{h \equiv h'}{ab \blacktriangleright h \equiv ab \blacktriangleright h'}$$

$$[\text{s-queue 1}] \quad \varnothing \cdot h \equiv h \qquad [\text{s-queue 2}] \quad h \cdot \varnothing \equiv h \qquad [\text{s-queue 3}] \quad h_1 \cdot (h_2 \cdot h_3) \equiv (h_1 \cdot h_2) \cdot h_3$$

Table 10: Structural congruence for asynchronous processes.

$$[\text{r-send-async}] \quad ab \blacktriangleright h \mid a!l\langle c\rangle.P \rightarrow ab \blacktriangleright h \cdot l\langle c\rangle \mid P \qquad [\text{r-receive-async}] \quad \frac{k \in I}{ab \blacktriangleright l_k\langle c\rangle \cdot h \mid \sum_{i \in I} b?l_i(x_i).P_i \rightarrow ab \blacktriangleright h \mid P_k\{{}^{c}/_{x_k}\}}$$

Table 11: Reduction of asynchronous processes.

4.2. **Errors in asynchronous processes.** Like in the synchronous case, also in the asynchronous case the errors arise when the duality of the communications offered by co-channels is lost. The presence of queues makes the definition of the error reductions for asynchronous communications not trivial. In addition to the other errors, we need to identify the following classical error situations (the terminology is adopted from works on Communicating Finite State Machines [12, 29, 64]):

(1) deadlocks: there are inputs waiting to dequeue messages from queues which will be forever empty.

(2) orphan message errors: there are messages in queues which will never be received by corresponding inputs, i.e. orphan messages will remain forever in queues.

Both errors are important, since we want to ensure every input can receive a message and every message in a queue can be read. These errors correspond to the following processes:

(1) deadlocks: a pair of co-channels are waiting for inputs and their queues are both empty, or one channel is waiting for an input over an empty queue and the co-channel only occurs as a queue name. In the first case the process has the shape

$$(\nu ab)(\sum_{i\in I} a?l_i(x_i).P_i \mid \sum_{j\in J} b?l'_j(x'_j).Q_j \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing)$$

and in the second case the process has the shape

$$(\nu ab)(\sum_{i\in I} a?l_i(x_i).P_i \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright h)$$

(2) orphan message errors: a queue is not empty, but the corresponding channel will never appear as subject of an input. I.e. the process has the shape $(\nu ab)(P \mid ba \blacktriangleright h)$ where h is not empty, but P will neither reduce to a process which performs an input on channel a, nor pass the channel a to an outer process.

To define statically the second error situation, we need to compute an over approximation (denoted by $\varphi(P)$) of the set of free channels which *might* eventually appear as subjects of *inputs* by *reducing* a process P. The definition of $\varphi(P)$ requires some care. We cannot simply modify the mapping $\mathsf{sc}(P)$ (page 6), since it only takes into account the subjects which occur free in P, without considering its unfoldings in case of recursion. Moreover, we cannot take the whole set of channels occurring free in P, because it is too large: for example, a recursive process which only sends messages will always contain both the channels of the subjects and of the objects of the outputs, but it will never read a message on its queue. We also need to carefully consider the reductions of P, as shown for example by the process:

$$P = b?l_0(x).x?l_1(y) \mid c!l_0\langle a\rangle \mid cb \blacktriangleright \varnothing \tag{4.1}$$

Notice that P does not contain inputs with subject a, but

$$P \to_{\mathsf{a}} b?l_0(x).x?l_1(y) \mid cb \blacktriangleright l_0\langle a\rangle \to_{\mathsf{a}} a?l_1(y) \mid cb \blacktriangleright \varnothing$$

and this last process has an input with subject a. Hence to define $\varphi(P)$, we need to take care of channels carried by outputs as well as channels occurring in messages inside queues. Another delicate case in the definition of $\varphi(P)$ comes from recursive definitions. For example the process

$$Q \;=\; \texttt{def}\ X(x) = a!l\langle x\rangle\ \texttt{in}\ X\langle c\rangle \mid b?l(y).y?l(z) \mid ab \blacktriangleright \varnothing \mid ba \blacktriangleright \varnothing$$

$$\delta(X\langle\tilde{u}\rangle,\tilde{D},\chi) \;=\; \begin{cases} \delta(P\{\tilde{u}/\tilde{x}\},\tilde{D},X\langle\tilde{u}\rangle\cdot\chi) & \text{if } X\langle\tilde{u}\rangle\notin\chi \text{ and} \\ & X(\tilde{x})=P\in\tilde{D} \\ \emptyset & \text{otherwise} \end{cases} \qquad \delta(\mathbf{0},\tilde{D},\chi) \;=\; \emptyset$$

$$\begin{array}{rclrcl} \delta(\sum_{i\in I} u?l_i(x_i).P_i,\tilde{D},\chi) &=& \{u\}\cup\bigcup_{i\in I}\delta(P_i,\tilde{D},\chi)\setminus\{x_i\} & \delta(u!l\langle u'\rangle.P,\tilde{D}) &=& \{u'\}\cup\delta(P,\tilde{D},\chi) \\ \delta(P_1\mid P_2,\tilde{D},\chi) &=& \delta(P_1,\tilde{D},\chi)\cup\delta(P_2,\tilde{D},\chi) & \delta(ab\blacktriangleright h,\tilde{D},\chi) &=& \gamma(h) \\ \delta(P_1\oplus P_2,\tilde{D},\chi) &=& \delta(P_1,\tilde{D},\chi)\cup\delta(P_2,\tilde{D},\chi) & \delta((\nu ab)P,\tilde{D},\chi) &=& \delta(P,\tilde{D},\chi)\setminus\{a,b\} \\ \delta(\texttt{def } D \texttt{ in } P,\tilde{D},\chi) &=& \delta(P,D\cdot\tilde{D},\chi) & & & \end{array}$$

Table 12: The mapping δ.

does not contain inputs with subject c, but

$$\begin{array}{rll} Q & \to_{\mathsf{a}} & \texttt{def } X(x)=a!l\langle x\rangle \texttt{ in } a!l\langle c\rangle \mid b?l(y).y?l(z)\mid ab\blacktriangleright\varnothing\mid ba\blacktriangleright\varnothing \\ & \to_{\mathsf{a}} & \texttt{def } X(x)=a!l\langle x\rangle \texttt{ in } b?l(y).y?l(z)\mid ab\blacktriangleright l\langle c\rangle\mid ba\blacktriangleright\varnothing \\ & \to_{\mathsf{a}} & \texttt{def } X(x)=a!l\langle x\rangle \texttt{ in } c?l(z)\mid ab\blacktriangleright\varnothing\mid ba\blacktriangleright\varnothing \qquad \text{(note the input with subject } c\text{)} \end{array}$$

In Definition 4.1 below, we introduce $\varphi(P)$ using two auxiliary mappings: γ and δ. Let $\tilde{D}$ range over sets of declarations and $D\cdot\tilde{D}$ denote the addition of D to the set $\tilde{D}$. Let χ range over sets of process variables and $X\langle\tilde{u}\rangle\cdot\chi$ denote the addition of $X\langle\tilde{u}\rangle$ to the set χ.

Definition 4.1. (Mappings γ,δ, and φ) The mapping γ is defined by induction on queues:

$$\gamma(\varnothing)=\emptyset \qquad \gamma(l\langle a\rangle)=\{a\} \qquad \gamma(h_1\cdot h_2)=\gamma(h_1)\cup\gamma(h_2)$$

The mapping δ is defined (by induction on processes) in Table 12, using γ. The mapping φ is defined as $\varphi(P)=\delta(P,\emptyset,\emptyset)$.

The mapping γ applied to a message queue h gives the set of channels which occur in h. The mapping δ has three arguments: a process P, a set of declarations and a set of process invocations. The result is the set of free channels which *might* occur as input subjects along the reductions of P: the declarations can be used in these reductions, and the set of process invocations memorises the process calls already considered. The mapping δ uses the mapping γ for dealing with queues. The mapping φ is the mapping δ applied to a process, an empty set of declarations, and an empty set of process invocations. Notably, if P only contains recursive process definitions $X(x)=P'$ (i.e., with one parameter x), and all recursive calls to X in P' have the form $X\langle x\rangle$ (i.e., each call to X reuses x), then $\varphi(P)=\delta(P,\emptyset,\emptyset)$ is well founded and each recursive call is unfolded only once (see Example 4.2). We will see that these properties are enjoyed by characteristic *asynchronous* processes, that will be introduced in Definition 5.1 for studying the preciseness of subtyping in the asynchronous calculus.

For example, for the process P shown in (4.1), we obtain $\varphi(P)=\{a,b\}$.

Example 4.2. *Assume*

$$P \;=\; \texttt{def } X(x)=a!l\langle x\rangle.X\langle x\rangle \texttt{ in } X\langle c\rangle\mid b?l(y).y?l(z)\mid ab\blacktriangleright\varnothing\mid ba\blacktriangleright\varnothing$$

$$\text{[ERR-MISM-ASYNC]}\quad \frac{\forall i \in I : l \neq l_i}{ab \blacktriangleright l\langle c\rangle \cdot h \mid \sum_{i\in I} b?l_i(x_i).P_i \to \texttt{error}}$$

$$\text{[ERR-IN-IN-ASYNC]}\quad (\nu ab)(\sum_{i\in I} a?l_i(x_i).P_i \mid \sum_{j\in J} b?l'_j(x'_j).Q_j \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing) \to \texttt{error}$$

$$\text{[ERR-IN-ASYNC]}\quad (\nu ab)(\sum_{i\in I} a?l_i(x_i).P_i \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright h) \to \texttt{error}$$

$$\text{[ERR-ORPH-MESS-ASYNC]}\quad \frac{a \notin \varphi(P) \quad \mathsf{fpv}(P) = \emptyset \quad h \neq \varnothing}{(\nu ab)(P \mid ba \blacktriangleright h) \to \texttt{error}}$$

Table 13: Error reduction for asynchronous processes.

Then we get:

$$\begin{array}{rcl}
\varphi(P) & = & \delta(P, \emptyset, \emptyset) \\
& = & \delta(X\langle c\rangle \mid b?l(y).y?l(z) \mid ab \blacktriangleright \varnothing \mid ba \blacktriangleright \varnothing, D, \emptyset) \\
& = & \delta(X\langle c\rangle, D, \emptyset) \cup \delta(b?l(y).y?l(z), D, \emptyset) \cup \delta(ab \blacktriangleright \varnothing, D, \emptyset) \cup \delta(ba \blacktriangleright \varnothing, D, \emptyset) \\
& = & \delta(a!l\langle c\rangle.X\langle c\rangle, D, X\langle c\rangle) \cup \{b\} \cup (\delta(y?l(z), D, \emptyset) \setminus \{y\}) \\
& = & \{c\} \cup \{b\} = \{c, b\}
\end{array}$$

where D is $X(x) = a!l\langle x\rangle.X\langle x\rangle$. The evaluation of $\varphi(P)$ tells that c can become the subject of an input. We illustrate this fact also by the following reduction:

$$\begin{array}{rl}
P & \to_{\mathsf{a}} \ \texttt{def}\ X(x) = a!l\langle x\rangle.X\langle x\rangle\ \texttt{in}\ a!l\langle c\rangle.X\langle c\rangle \mid b?l(y).y?l(z) \mid ab \blacktriangleright \varnothing \mid ba \blacktriangleright \varnothing \\
& \to_{\mathsf{a}} \ \texttt{def}\ X(x) = a!l\langle x\rangle.X\langle x\rangle\ \texttt{in}\ X\langle c\rangle \mid b?l(y).y?l(z) \mid ab \blacktriangleright l\langle c\rangle \mid ba \blacktriangleright \varnothing \\
& \to_{\mathsf{a}} \ \texttt{def}\ X(x) = a!l\langle x\rangle.X\langle x\rangle\ \texttt{in}\ X\langle c\rangle \mid c?l(z) \mid ab \blacktriangleright \varnothing \mid ba \blacktriangleright \varnothing
\end{array}$$

The error reduction rules for asynchronous processes are [ERR-CONTEXT] of Table 4 and the rules of Table 13. Rule [ERR-MISM-ASYNC] deals with a label mismatch between a message on the top of the queue and an input. Rule [ERR-IN-IN-ASYNC] gives an error when two processes with restricted channels are in deadlock waiting to read from empty queues. Rule [ERR-IN-ASYNC] deals with the case of one process waiting to read from an empty queue which will never contain a message, since there are no occurrences of the unique channel that can enqueue messages. Rule [ERR-ORPH-MESS-ASYNC] corresponds to the orphan message error. Clearly $(\nu ab)(P \mid ba \blacktriangleright h)$ could reduce in other ways, take for example $P = b!l\langle c\rangle$. In this rule the condition $\mathsf{fpv}(P) = \emptyset$ assures that we consider all needed declarations in computing $\varphi(P)$.

Definition 4.3. We write $P \to_{\mathsf{a}} \texttt{error}$ if $P \to \texttt{error}$ can be derived using the rules of Table 13 plus rule [ERR-CONTEXT] of Table 4. The notation $\to_{\mathsf{a}}^*$ is used with the expected meaning.

Let $\boldsymbol{P}(a, T)$ range over characteristic asynchronous processes as in Definition 5.1. We will use the definitions above to show that a typeable process $\boldsymbol{P}(a, T) \mid \boldsymbol{P}(b, S) \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing$ does not reduce to `error`. *Vice versa*, we will also show that if the process $\boldsymbol{P}(a, T) \mid \boldsymbol{P}(b, S) \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing$ cannot be typed, then there are $T' \leqslant_{\mathsf{s}} T$ and $S' \leqslant_{\mathsf{s}} S$ such that $\boldsymbol{P}(a, T') \mid \boldsymbol{P}(b, S') \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing$ reduces to `error`. As in the synchronous case, arbitrary

processes can be stuck without reducing to `error`: this happens, for instance, if a process inputs/outputs on a channel without a corresponding queue. Note that the type system we will introduce in § 4.4 is an adaptation of those by Mostrous *et al.* [49, 51] to our calculus, and they do not aim at avoiding deadlock or orphan message errors.

Similarly to Proposition 2.1, if an asynchronous process can reduce to `error` in one step, then a different reduction produces a process with the same property.

Proposition 4.4. If $P \to_{\mathsf{a}} \texttt{error}$ and $P \to_{\mathsf{a}} P' \neq \texttt{error}$, then $P' \to_{\mathsf{a}} \texttt{error}$.

Proof. By cases on the rule giving $P \to_{\mathsf{a}} \texttt{error}$ (Definition 4.3). The statement holds vacuously for [ERR-MISM-ASYNC], [ERR-IN-IN-ASYNC] and [ERR-IN-ASYNC] (Table 13): in such cases, $P \to_{\mathsf{a}} P'$ implies $P' = \texttt{error}$.

In the case [ERR-ORPH-MESS-ASYNC], we have $P = (\nu ab)(Q \mid ba \blacktriangleright h) \to_{\mathsf{a}} \texttt{error}$; moreover, $a \notin \varphi(Q)$, $\mathsf{fpv}(Q) = \emptyset$ and $h \neq \varnothing$. If $P \to_{\mathsf{a}} P' \neq \texttt{error}$, then (by [R-CONTEXT]) it must be $P' = (\nu ab)(Q' \mid ba \blacktriangleright h')$ with $Q \mid ba \blacktriangleright h \to_{\mathsf{a}} Q' \mid ba \blacktriangleright h'$: by induction on the derivation of this transition, we can verify that $a \notin \varphi(Q')$, $\mathsf{fpv}(Q') = \emptyset$ and $h' \neq \varnothing$. Hence, again by rule [ERR-ORPH-MESS-ASYNC], we conclude $P' \to_{\mathsf{a}} \texttt{error}$.

The case [ERR-CONTEXT] (Table 4) is proved similarly to Proposition 2.1. □

4.3. **Asynchronous subtyping.** The asynchronous subtyping is not only essential for the completeness result, but it is also important in practice. As observed by Yoshida, Honda *et al.* [52, 67], implementing this subtyping is a key tool for maximising message-overlapping in the high-performance computing environments. To explain the usefulness of the asynchronous subtyping, consider:

$$P_1 = a?l(y_1).a!l\langle 5\rangle.Q_1 \quad P_2 = b!l\langle Large\_datum\rangle.b?l(y_2).Q_2$$

First P_2 sends a large datum on channel b; then after receiving it, P_1 sends 5 to P_2. We note that P_1's output is blocked until this large datum is received. Since the value replacing y_1 does not influence the subsequent output at a, process P_1 can be optimised by sending the small datum "5" first, so that once the large datum is put in the queue at ba, process P_2 can immediately receive the small datum. Thus a better version of P_1 is P_1' defined by:

$$P_1' = a!l\langle 5\rangle.a?l(y_1).Q_1$$

Asynchronous subtyping specifies safe permutations of actions, by which we can refine a local protocol to maximise asynchrony without violating session safety.

To define asynchronous subtyping, the notion of asynchronous context of types is handy [48, 50]. An asynchronous context is a sequence of branchings containing holes that we index in order to distinguish them.

Definition 4.5 (Asynchronous context)**.**

$$\mathcal{A} \;::=\; [\,]^n \;\mid\; \&_{i\in I}?l_i(S_i).\mathcal{A}_i$$

We write $\mathcal{A}[\,]^{n\in N}$ to denote a context with holes indexed by elements of N and $\mathcal{A}[T_n]^{n\in N}$ to denote the same context when the hole $[\,]^n$ has been filled with T_n.

We naturally extend the definition of type tree (page 9) to contexts, letting $\mathcal{T}([\,]^n) = [\,]^n$. For an example, see Figure 2. Note that each continuation path of a context tree is *finite*, only connects branching nodes, and ends on a hole.

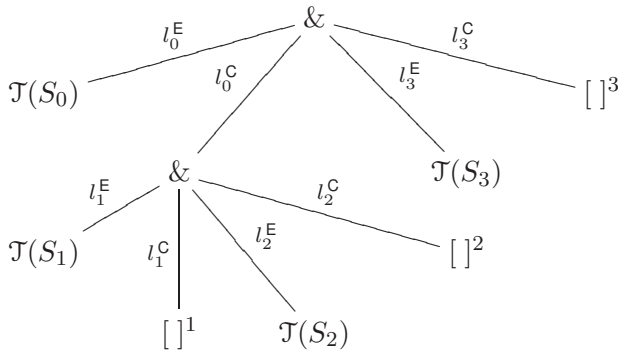


Figure 2: The tree of $\mathcal{A} = ?l_0(S_0).\left(?l_1(S_1).[\,]^1 \,\&\, ?l_2(S_2).[\,]^2\right) \,\&\, ?l_3(S_3).[\,]^3$.

Example 4.6. *Let* $N = \{1, 2\}$ *and*

$$T_1 = !\mathsf{m}\langle S_m\rangle.T_m \oplus !\mathsf{p}\langle S_p\rangle.T_p, \qquad T_2 = !\mathsf{m}\langle S'_m\rangle.T'_m \oplus !\mathsf{p}\langle S'_p\rangle.T'_p \oplus !\mathsf{q}\langle S_q\rangle.T_q.$$

Assume $\mathcal{A} = ?\mathsf{r}(S_r).[\,]^1 \,\&\, ?\mathsf{s}(S_s).[\,]^2$, *then*

$$\mathcal{A}[T_1]^1[T_2]^2 = ?\mathsf{r}(S_r).(!\mathsf{m}\langle S_m\rangle.T_m \oplus !\mathsf{p}\langle S_p\rangle.T_p) \,\&\, ?\mathsf{s}(S_s).(!\mathsf{m}\langle S'_m\rangle.T'_m \oplus !\mathsf{p}\langle S'_p\rangle.T'_p \oplus !\mathsf{q}\langle S_q\rangle.T_q).$$

To define asynchronous subtyping we also need to introduce the predicate $\& \in T$, which holds if each continuation path of $\mathcal{T}(T)$ contains at least one branching. The regularity of the trees corresponding to session types assures that the branchings occur at finite levels. More precisely $\& \in T$ holds if we can derive it from the following axioms and inductive rules:

$$\& \in \&_{i\in I} ?l_i(S_i).T_i \qquad \frac{\forall i \in I \quad \& \in T_i}{\& \in \bigoplus_{i\in I} !l_i\langle S_i\rangle.T_i} \qquad \frac{\& \in T}{\& \in \mu\mathbf{t}.T}$$

where we consider also open session types; note that such a predicate holds up-to unfolding, i.e. $\& \in \mu\mathbf{t}.T$ iff $\& \in T\{^{\mu\mathbf{t}.T}/_{\mathbf{t}}\}$ (Proposition B.1 in Appendix B). We write $\& \in \mathcal{A}$ if $\mathcal{A}$ is a branching, i.e. it is not a single hole.

Example 4.7. *We have:*

$$\& \in !l\langle S\rangle.?l'(S').\mathtt{end} \quad \& \in \mu\mathbf{t}.!l\langle S\rangle.?l'(S').\mathbf{t} \quad \neg\left(\& \in \mu\mathbf{t}.\left(!l\langle S\rangle.?l'(S').\mathbf{t} \oplus !l''\langle S''\rangle.\mathbf{t}\right)\right)$$

In the first and second case, the tree of each type has just one continuation path, with a selection followed by a branching, and we have corresponding (finite) derivations under the rules above. In the third case, the recursion guarded by the l''*-labelled output generates an* infinite *continuation path without branchings: this yields an infinite (and thus, invalid) derivation under the inductive rules for* $\& \in T$*; since no finite derivations exist, the predicate does not hold.*

We can now define the asynchronous subtyping relation $T \leqslant_{\mathsf{a}} S$, which holds if $T \leqslant S$ is derived by the rule:

$$\text{[SUB-PERM-ASYNC]}\quad \frac{\forall i \in I\ \forall n \in N: \quad S_i^n \leqslant S_i \quad T_i \leqslant \mathcal{A}[T_i^n]^{n\in N} \quad \& \in \mathcal{A} \quad \& \in T_i}{\bigoplus_{i\in I} !l_i\langle S_i\rangle.T_i \leqslant \mathcal{A}[\bigoplus_{i\in I\cup J_n} !l_i\langle S_i^n\rangle.T_i^n]^{n\in N}}$$

$$\frac{S^r_m \leqslant S_m \quad S^s_m \leqslant S_m \quad S^r_p \leqslant S_p \quad S^s_p \leqslant S_p \quad T_m \leqslant ?\mathsf{r}(S_r).T_r \,\&\, ?\mathsf{s}(S_s).T_s \quad T_p \leqslant ?\mathsf{r}(S_r).T'_r \,\&\, ?\mathsf{s}(S_s).T'_s}{!\mathsf{m}\langle S_m\rangle.T_m \oplus !\mathsf{p}\langle S_p\rangle.T_p \;\leqslant\; ?\mathsf{r}(S_r).(!\mathsf{m}\langle S^r_m\rangle.T_r \oplus !\mathsf{p}\langle S^r_p\rangle.T'_r \oplus !\mathsf{q}\langle S_q\rangle.T_q) \,\&\, ?\mathsf{s}(S_s).(!\mathsf{m}\langle S^s_m\rangle.T_s \oplus !\mathsf{p}\langle S^s_p\rangle.T'_s)}$$

Figure 3: Application of [SUB-PERM-ASYNC], where $T_m = ?\mathsf{r}(S_r).T_r \,\&\, ?\mathsf{s}(S_s).T_s \,\&\, ?\mathsf{u}(S_u).T_u$ and $T_p = ?\mathsf{r}(S'_r).T'_r \,\&\, ?\mathsf{s}(S_s).T'_s$ and we assume $S'_r \leqslant S_r$.

together with the rules in Table 6.

Rule [SUB-PERM-ASYNC] allows the asynchronous safe permutation explained above. It postpones a selection after an unbounded but finite number of branchings, and the selections inside these branchings can be bigger according to rule [SUB-SEL] of Table 6. The conditions $\& \in \mathcal{A}$ and $\& \in T_i$ for all $i \in I$ are crucial for the soundness of this rule. Without these conditions we get the subtyping of Mostrous PhD thesis [48, 50] (for further discussion on this topic, see § 8, paragraph "Subtyping of Mostrous PhD thesis").

Reflexivity of $\leqslant_{\mathsf{a}}$ is immediate, while the proof of transitivity requires some ingenuity, see Appendix B.

Theorem 4.8. *The relation $\leqslant_{\mathsf{a}}$ is transitive.*

Example 4.9. (Asynchronous subtyping)

(1) *We show $T_1 \leqslant_{\mathsf{a}} S_1$, where $T_1 = \mu\mathbf{t}.!l\langle T'\rangle.?l'(S').\mathbf{t}$ and $S_1 = \mu\mathbf{t}.?l'(S').!l\langle T'\rangle.\mathbf{t}$. If we assume $T_1 \leqslant S_1$, we obtain*

$$!l\langle T'\rangle.?l'(S').T_1 \leqslant ?l'(S').!l\langle T'\rangle.S_1$$

by rule [SUB-PERM-ASYNC], which is $T_1 \leqslant S_1$ by folding. In this way we coinductively proved $T_1 \leqslant_{\mathsf{a}} S_1$.

(2) *We show $T_2 \leqslant_{\mathsf{a}} S_2$, where $T_2 = !l\langle T'\rangle.T_1$ and $S_2 = ?l'(S').S_1$ and T_1, S_1 are as in previous example. We assume $T_2 \leqslant S_2$. We get*

$$\begin{array}{rcll} !l\langle T'\rangle.!l\langle T'\rangle.?l'(S').T_1 & \leqslant & ?l'(S').!l\langle T'\rangle.!l\langle T'\rangle.T_1 & \text{by rule [SUB-PERM-ASYNC]} \\ & \leqslant & ?l'(S').!l\langle T'\rangle.?l'(S').S_1 & \text{by the assumption } T_2 \leqslant S_2 \\ & \leqslant & ?l'(S').?l'(S').!l\langle T'\rangle.S_1 & \text{by rule [SUB-PERM-ASYNC]} \end{array}$$

which is $T_2 \leqslant S_2$ by folding. In this way we proved $T_2 \leqslant_{\mathsf{a}} S_2$ using transitivity.

(3) *Choosing $\mathcal{A}$ as in Example 4.6, Figure 3 gives an application of rule [SUB-PERM-ASYNC]. The rightmost premises can be written*

$$T_m \leqslant \mathcal{A}[T_r]^1[T_s]^2 \text{ and } T_p \leqslant \mathcal{A}[T'_r]^1[T'_s]^2$$

and they hold by rule [SUB-BRA]. The left-hand-side of the conclusion is a selection between the outputs $!\mathsf{m}\langle S_m\rangle.T_m$ and $!\mathsf{p}\langle S_p\rangle.T_p$. The right-hand-side of the conclusion can be written

$$\mathcal{A}[!\mathsf{m}\langle S^r_m\rangle.T_r \oplus !\mathsf{p}\langle S^r_p\rangle.T'_r \oplus !\mathsf{q}\langle S_q\rangle.T_q]^1[!\mathsf{m}\langle S^s_m\rangle.T_s \oplus !\mathsf{p}\langle S^s_p\rangle.T'_s]^2$$

Notice that selections are moved inside branching (possibly making smaller the types of the sent channels) and extra selections (in this case $!\mathsf{q}\langle S_q\rangle.T_q$) can be added.

By definition $\leqslant_{\mathsf{a}}$ is an extension of $\leqslant_{\mathsf{s}}$, and the example at the beginning of this subsection shows that $\leqslant_{\mathsf{s}}$ is not complete for the asynchronous calculus.

$$[\text{T-NEW-ASYNC}]\quad \frac{\Gamma \vdash P \triangleright \Delta, a : T_1, b : T_2, ba : \tau_1, ab : \tau_2 \quad T_1 - \tau_1 \bowtie T_2 - \tau_2}{\Gamma \vdash (\nu ab)P \triangleright \Delta}$$

$$[\text{T-EMPTY-Q}]\quad \Gamma \vdash ba \blacktriangleright \varnothing \triangleright ba : \epsilon$$

$$[\text{T-MESSAGE-Q}]\quad \frac{\Gamma \vdash ba \blacktriangleright h \triangleright \Delta, ba : \tau}{\Gamma \vdash ba \blacktriangleright h \cdot l\langle c\rangle \triangleright \Delta, c : S, ba : \tau \cdot l\langle S\rangle}$$

Table 14: Typing rules for asynchronous processes and queues.

4.4. **Typing asynchronous processes.** Since processes now include queues, we need queue types defined by:

$$\tau \quad ::= \quad \epsilon \mid l\langle S\rangle \mid \tau \cdot \tau$$

where we assume associativity of $\cdot$ and $\tau \cdot \epsilon = \epsilon \cdot \tau = \tau$. We also extend session environments as follows:

$$\Delta \quad ::= \quad \ldots \mid \Delta, ab : \tau$$

The added new element $ab : \tau$ is the type of messages in the queue $ab \blacktriangleright h$.

We denote by $\mathsf{dom_q}(\Delta)$ the set of local queues which occur in Δ. Two session environments Δ_1 and Δ_2 *agree* if

$$\mathsf{dom}(\Delta_1) \cap \mathsf{dom}(\Delta_2) = \mathsf{dom_q}(\Delta_1) \cap \mathsf{dom_q}(\Delta_2) = \emptyset.$$

If Δ_1 and Δ_2 agree, their composition Δ_1, Δ_2 is given by $\Delta_1, \Delta_2 = \Delta_1 \cup \Delta_2$ as in the synchronous case. We also define $\Delta_1 \leqslant_{\mathsf{a}} \Delta_2$ by:

$$\begin{array}{l} u \in \mathsf{dom}(\Delta_1) \cap \mathsf{dom}(\Delta_2) \text{ implies } \Delta_1(u) \leqslant_{\mathsf{a}} \Delta_2(u) \text{ and} \\ u \in \mathsf{dom}(\Delta_1) \text{ and } u \notin \mathsf{dom}(\Delta_2) \text{ imply } \Delta_1(u) = \mathtt{end} \text{ and} \\ u \notin \mathsf{dom}(\Delta_1) \text{ and } u \in \mathsf{dom}(\Delta_2) \text{ imply } \Delta_2(u) = \mathtt{end} \text{ and} \\ \mathsf{dom_q}(\Delta_1) = \mathsf{dom_q}(\Delta_2) \text{ and } ab \in \mathsf{dom_q}(\Delta_1) \text{ implies } \Delta_1(ab) = \Delta_2(ab) \end{array}$$

We write $\Delta_1 \approx_{\mathsf{a}} \Delta_2$ if $\Delta_1 \leqslant_{\mathsf{a}} \Delta_2$ and $\Delta_2 \leqslant_{\mathsf{a}} \Delta_1$.

We need to take into account the interplay between the session type of a channel and the queue type of the queue dequeued by this channel. Following the literature [25, 48, 49, 50] we define the *session remainder* of a session type T and a queue type τ (notation $T - \tau$). The session type $T - \tau$ is obtained from T by erasing all branchings in T that have corresponding selections in τ. Clearly the session remainder is defined only if T and τ agree on labels and on types of exchanged channels. More formally we define:

$$[\text{RM-EMPTY}]\quad T - \epsilon = T$$

$$[\text{RM-BRA}]\quad \frac{T_k - \tau = T' \quad S_k \leqslant_{\mathsf{a}} S \quad k \in I}{\&_{i\in I} ?l_i(S_i).T_i - l_k\langle S\rangle \cdot \tau = T'}$$

$$[\text{RM-SEL}]\quad \frac{\forall i \in I : T_i - \tau = T_i'}{\bigoplus_{i\in I} !l_i\langle S_i\rangle.T_i - \tau = \bigoplus_{i\in I} !l_i\langle S_i\rangle.T_i'}$$

The typing rules for asynchronous processes are obtained from the rules of Table 7 by replacing rule [T-NEW-SYNC] with rule [T-NEW-ASYNC] and $\leqslant_{\mathsf{s}}$ with $\leqslant_{\mathsf{a}}$ in rule [T-SUB] and adding the rules for typing the queues. Table 14 gives all the new rules. In rule [T-NEW-ASYNC] we take into account not only the types of the channels, but also those of the queues, and we require duality between their remainders. Rule [T-EMPTY-Q] types the empty queue. Rule [T-MESSAGE-Q] says how the type of a queue changes when one message is added.

$$[\text{TR-IN}]\quad \frac{S_k \leqslant_{\mathsf{a}} S \qquad k \in I}{b : \&_{i\in I} ?l_i(S_i).T_i, ab : l_k\langle S\rangle \cdot \tau \Rightarrow b : T_k, ab : \tau} \qquad [\text{TR-RES}]\quad \frac{\Delta_1 \Rightarrow \Delta_1'}{\Delta_1, \Delta_2 \Rightarrow \Delta_1', \Delta_2}$$

$$[\text{TR-OUT}]\quad \frac{\forall n \in N\ \exists i_n \in I_n :\ l^n_{i_n} = l \quad S^n_{i_n} \leqslant_{\mathsf{a}} S}{a : \mathcal{A}[\bigoplus_{i\in I_n} !l^n_i\langle S^n_i\rangle.T^n_i]^{n\in N}, ab : \tau \Rightarrow a : \mathcal{A}[T^n_{i_n}]^{n\in N}, ab : \tau \cdot l\langle S\rangle}$$

Table 15: Reduction of asynchronous session environments.

4.5. **Soundness of asynchronous subtyping.** Reduction of session environments is standard in session calculi, to take into account how communications modify the types of free channels and queues [7, 36, 66]. In the synchronous case only restricted channels can exchange messages. We could reduce only restricted channels also in the asynchronous case, but this would make the reduction rules heavier. Table 15 defines the reduction between session environments. Rule [TR-IN] simply corresponds to the dequeue of a message. Rule [TR-OUT] takes into account the asynchronous subtyping: we need to choose one type out of a selection under a context, since a typeable process might contain a selection followed by several branches, thanks to rule [SUB-PERM-ASYNC]. The following example illustrates rule [TR-OUT].

Example 4.10. *Let $T_1, T_2, \mathcal{A}$ be defined as in Example 4.6 and assume that there is S such that $S_m, S'_m \leqslant_{\mathsf{a}} S$ and that there is no S' such that $S_p, S'_p \leqslant_{\mathsf{a}} S'$. By rule [TR-OUT], only branch m can be triggered to output, since branch q is only at hole 2, and the above assumption forbids to choose branch p.*

$$a : \mathcal{A}[T_1]^1[T_2]^2, ab : \tau \Rightarrow a : ?\mathsf{r}(S_r).T_m \ \&\ ?\mathsf{s}(S_s).T'_m, ab : \tau \cdot \mathsf{m}\langle S\rangle$$

In order to get subject reduction we cannot start from an arbitrary session environment. For example the process

$$a?l(x).a!l'\langle x+1\rangle \mid ba \blacktriangleright l\langle \text{true}\rangle$$

can be typed with the session environment

$$\{a : ?l(\texttt{int}).!l'\langle\texttt{int}\rangle.\texttt{end}, ba : l\langle\texttt{bool}\rangle\}$$

but it reduces to $a!l'\langle \text{true} + 1\rangle$ which cannot be typed. As usual [7, 36, 66] we restrict to balanced session environments according to the following definition.

Definition 4.11 (Balanced session environments)**.** A session environment Δ is *balanced* if:
(1) $a : T, ba : \tau \in \Delta$ imply that $T - \tau$ is defined; and
(2) $a : T, ba : \tau, b : T', ab : \tau' \in \Delta$ imply that $T - \tau \bowtie T' - \tau'$.

Notice that $?l(\texttt{int}).!l'\langle\texttt{int}\rangle.\texttt{end} - l\langle\texttt{bool}\rangle$ is undefined, if we extend in the obvious way the definition of session remainder.

It is easy to verify that reduction preserves balancing of session environments.

Lemma 4.12. *If $\Delta \Rightarrow \Delta'$ and Δ is balanced, then Δ' is balanced.*

We can now state subject reduction:

Theorem 4.13 (Subject reduction for asynchronous processes)**.** *If* $\Gamma \vdash_{\mathsf{a}} P \triangleright \Delta$ *and* Δ *is balanced and* $P \rightarrow_{\mathsf{a}}^{*} Q$*, then there is* Δ' *such that* $\Delta \Rightarrow^{*} \Delta'$ *and* $\Gamma \vdash_{\mathsf{a}} Q \triangleright \Delta'$*.*

Also the assurance that well-typed processes cannot go wrong requires balanced session environments.

Corollary 4.14. *If* $\Gamma \vdash_{\mathsf{a}} P \triangleright \Delta$ *and* Δ *is balanced, then* $P \not\rightarrow_{\mathsf{a}}^{*} \texttt{error}$*.*

Proof. By Theorem 4.13, $P \rightarrow_{\mathsf{a}}^{*} \texttt{error}$ implies $\Gamma \vdash_{\mathsf{a}} \texttt{error} \triangleright \Delta'$ for some Δ', which is impossible because `error` has no type. □

Lastly we get:

Theorem 4.15. *The asynchronous subtyping relation* $\leqslant_{\mathsf{a}}$ *is sound for the asynchronous calculus.*

The proofs of Theorems 4.13 and 4.15 are given in Appendix B.

5. Completeness for Asynchronous Subtyping

We start this section by remarking that the synchronous subtyping is incomplete for the asynchronous calculus. In fact the synchronous subtyping is strictly included in the asynchronous subtyping. For example $T \not\leqslant_{\mathsf{s}} S$ by rule [N-SELBRA-SYNC] but $T \leqslant_{\mathsf{a}} S$, where $T = !l\langle T'\rangle.?l'(S')$ and $S = ?l'(S').!l\langle T'\rangle$. Then soundness of the asynchronous subtyping implies incompleteness of the synchronous subtyping.

We show completeness for asynchronous subtyping following the three steps described in § 3. In the third step we need to add two queues for exchanging messages, see the proof of Theorem 5.7. The proofs are more tricky than in the synchronous case since the asynchronous subtyping makes the shapes of types less structured. The first difficulty is to define the negation $\not\leqslant_{\mathsf{a}}$ inductively and prove that it implies the non-derivability of $\leqslant_{\mathsf{a}}$. The second difficulty is to catch the error states arising after an unbounded number of message enqueues, since rule [SUB-PERM-ASYNC] can exchange a selection with an unbounded number of branchings.

Characteristic asynchronous processes. The definition of characteristic processes for the asynchronous case differs from that of the synchronous one only for outputs, since the creation of a new pair of restricted channels requires also the creation of the corresponding queues.

Definition 5.1 (Characteristic asynchronous processes)**.** The characteristic process offering communication T on identifier u for the asynchronous calculus, denoted by $\boldsymbol{P}(u, T)$, is defined as in Definition 3.1, but for the case of $\boldsymbol{P}_!(u, l, S, T)$, which is now:

$$\boldsymbol{P}_!(u, l, S, T) \stackrel{\text{def}}{=} (\nu ab)(u!l\langle a\rangle.\boldsymbol{P}(u, T) \mid \boldsymbol{P}(b, \overline{S}) \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing)$$

For example if $T = !l_1\langle \mathtt{end}\rangle \oplus !l_2\langle !l_3\langle \mathtt{end}\rangle.\mathtt{end}\rangle.\mathtt{end}$, then

$$\begin{array}{rcl}
\boldsymbol{P}(a,T) & = & \boldsymbol{P}_!(a,l_1,\mathtt{end},\mathtt{end}) \oplus \boldsymbol{P}_!(a,l_2,!l_3\langle\mathtt{end}\rangle.\mathtt{end},\mathtt{end}) \\
& = & (\nu bb')(a!l_1\langle b\rangle.\boldsymbol{P}(a,\mathtt{end}) \mid \boldsymbol{P}(b',\mathtt{end}) \mid bb' \blacktriangleright \varnothing \mid b'b \blacktriangleright \varnothing) \oplus \\
& & (\nu cc')(a!l_2\langle c\rangle.\boldsymbol{P}(a,\mathtt{end}) \mid \boldsymbol{P}(c',?l_3(\mathtt{end}).\mathtt{end}) \mid cc' \blacktriangleright \varnothing \mid c'c \blacktriangleright \varnothing) \\
& = & (\nu bb')(a!l_1\langle b\rangle.\mathbf{0} \mid \mathbf{0} \mid bb' \blacktriangleright \varnothing \mid b'b \blacktriangleright \varnothing) \oplus \\
& & (\nu cc')(a!l_2\langle c\rangle.\mathbf{0} \mid c'?l_3(x).(\boldsymbol{P}(c',\mathtt{end}) \mid \boldsymbol{P}(x,\mathtt{end})) \mid cc' \blacktriangleright \varnothing \mid c'c \blacktriangleright \varnothing) \\
& = & (\nu bb')(a!l_1\langle b\rangle.\mathbf{0} \mid \mathbf{0} \mid bb' \blacktriangleright \varnothing \mid b'b \blacktriangleright \varnothing) \oplus \\
& & (\nu cc')(a!l_2\langle c\rangle.\mathbf{0} \mid c'?l_3(x).(\mathbf{0} \mid \mathbf{0}) \mid cc' \blacktriangleright \varnothing \mid c'c \blacktriangleright \varnothing) \\
& \equiv & (\nu bb')(a!l_1\langle b\rangle \mid bb' \blacktriangleright \varnothing \mid b'b \blacktriangleright \varnothing) \oplus (\nu cc')(a!l_2\langle c\rangle \mid c'?l_3(x) \mid cc' \blacktriangleright \varnothing \mid c'c \blacktriangleright \varnothing)
\end{array}$$

Similarly to Lemma 3.2 we get:

Lemma 5.2. $\vdash_{\mathsf{a}} \boldsymbol{P}(u,T) \triangleright \{u : T\}$.

Rules for negation of the asynchronous subtyping. The negation rules of asynchronous subtyping are obtained from the rules of Table 8 *excluding* rule [N-SELBRA-SYNC] plus the rules of Table 16. Rule [N-LABEL-ASYNC] deals with the case that the selection cannot find a matching label inside the n_0-th hole. Rule [N-EXCH-ASYNC] considers a mismatch between carried types inside the n_0-th hole. Rule [N-CONT-ASYNC] considers a mismatch between continuation types, again inside the n_0-th hole. The asynchronous context in these rules allows to consider selection surrounded by branchings. These three rules become the rules [N-LABEL-SEL], [N-EXCH-SEL] and [N-CONT-SEL] of Table 8, respectively, when the context $\mathcal{A}$ is just one hole. Rule [N-BRA-ASYNC] assures that a type without branchings cannot be a subtype of a branching type; the predicate $\& \notin T$ is the negation of the predicate $\& \in T$ (see page 22). More precisely $\& \notin T$ holds iff we can derive it from the following axioms

$$\text{[N-LABEL-ASYNC]}\quad \frac{\exists i_0 \in I\ \exists n_0 \in N\ \forall j \in J_{n_0} : l_j^{n_0} \neq l_{i_0}}{\bigoplus_{i\in I} !l_i\langle S_i\rangle.T_i \not\leftrightarrows \mathcal{A}[\bigoplus_{j\in J_n} !l_j^n\langle S_j^n\rangle.T_j^n]^{n\in N}}$$

$$\text{[N-EXCH-ASYNC]}\quad \frac{\exists i_0 \in I\ \exists n_0 \in N\ \exists j_0 \in J_{n_0} : l_{j_0}^{n_0} = l_{i_0} \quad S_{j_0}^{n_0} \not\leftrightarrows S_{i_0}}{\bigoplus_{i\in I} !l_i\langle S_i\rangle.T_i \not\leftrightarrows \mathcal{A}[\bigoplus_{j\in J_n} !l_j^n\langle S_j^n\rangle.T_j^n]^{n\in N}}$$

$$\text{[N-CONT-ASYNC]}\quad \frac{\forall i \in I\ \forall n \in N\ \exists j_{i,n} \in J_n : l_{j_{i,n}}^n = l_i \quad \exists i_0 \in I\ T_{i_0} \not\leftrightarrows \mathcal{A}[T_{j_{i_0,n}}^n]^{n\in N}}{\bigoplus_{i\in I} !l_i\langle S_i\rangle.T_i \not\leftrightarrows \mathcal{A}[\bigoplus_{j\in J_n} !l_j^n\langle S_j^n\rangle.T_j^n]^{n\in N}}$$

$$\text{[N-BRA-ASYNC]}\quad \frac{\& \notin T}{T \not\leftrightarrows \&_{i\in I} ?l_i(S_i).T_i} \qquad \text{[N-SEL-ASYNC]}\quad \frac{\oplus \notin T}{\bigoplus_{i\in I} !l_i\langle S_i\rangle.T_i \not\leftrightarrows T}$$

Table 16: Negation of asynchronous subtyping.

and inductive rules:

$$\& \notin \mathtt{end} \qquad \& \notin \mathbf{t} \qquad \frac{\exists i \in I \quad \& \notin T_i}{\& \notin \bigoplus_{i \in I} !l_i\langle S_i\rangle.T_i} \qquad \frac{\& \notin T}{\& \notin \mu \mathbf{t}.T}$$

where we consider also open session types; such a predicate holds up-to unfolding and is the complement of $\& \in T$ (Propositions C.1 and C.2 in Appendix C). Dually, rule [N-SEL-ASYNC] assures that a type without selections cannot be a subtype of a selection type. The predicate $\oplus \in T$ holds if each continuation path of $\mathfrak{T}(T)$ contains at least one selection and the predicate $\oplus \notin T$ is the negation of the predicate $\oplus \in T$. The definitions of these predicates are analogous to those of $\& \in T$ and $\& \notin T$. We write $T \not\lhd_{\mathsf{a}} S$ if $T \not\lhd S$ is generated by the rules in Table 16 and Table 8 excluding rule [N-SELBRA-SYNC].

For example, by rule [N-CONT-ASYNC],

$$!\mathsf{m}\langle S_m\rangle.(?\mathsf{r}(S_r).T_r \& ?\mathsf{s}(S_s).\mathtt{end}) \not\lhd_{\mathsf{a}} ?\mathsf{r}(S_r).!\mathsf{m}\langle S_m\rangle.T_r \& ?\mathsf{s}(S_s).!\mathsf{m}\langle S_m\rangle.!\mathsf{p}\langle S_p\rangle.T_p$$

since $?\mathsf{r}(S_r).T_r \& ?\mathsf{s}(S_s).\mathtt{end} \not\lhd_{\mathsf{a}} ?\mathsf{r}(S_r).T_r \& ?\mathsf{s}(S_s).!\mathsf{p}\langle S_p\rangle.T_p$.

In Lemma 5.3, we show that $\not\lhd_{\mathsf{a}}$ is the negation of the asynchronous subtyping. This result will be used (in the *"only if"* direction) in the proof of Theorem 5.7.

Lemma 5.3. *$T \leqslant_{\mathsf{a}} S$ is not derivable if and only if $T \not\lhd_{\mathsf{a}} S$ is derivable.*

Proof. If $T \not\lhd_{\mathsf{a}} S$, then we can show $T \not\leqslant_{\mathsf{a}} S$ by induction on the derivation of $T \not\lhd_{\mathsf{a}} S$. The proof is similar to that for "$T \not\lhd_{\mathsf{s}} S$ implies $T \not\leqslant_{\mathsf{s}} S$" (Lemma 3.3) — removing the base case [N-SELBRA-SYNC], and adding the cases for the rules in Table 16. We can further remove the cases [N-LABEL-SEL], [N-EXCH-SEL] and [N-CONT-SEL], since (as discussed above) they are subsumed respectively by [N-LABEL-ASYNC], [N-EXCH-ASYNC] and [N-CONT-ASYNC], when $\mathcal{A}$ is just one hole. We show the detailed proofs for some of the new cases (the omitted proofs are similar):

- base case [N-BRA-ASYNC]. We have $T \not\lhd S = \&_{i \in I} ?l_i(S_i).T_i$, with $\& \notin T$. Since S is a branching, $S \leqslant_{\mathsf{a}} T$ cannot hold by [SUB-END] nor [SUB-SEL]; moreover, it cannot hold by [SUB-BRA], since T cannot be a branching (otherwise, we would have the contradiction $\& \in T$); also [SUB-PERM-ASYNC] is ruled out: otherwise, we would have $T = \bigoplus_{i \in J} !l_i\langle S_i\rangle.T_i$ and $\forall i \in J : \& \in T_i$, and thus the contradiction $\& \in T$. Hence, we conclude $S \not\leqslant_{\mathsf{a}} T$;
- base case [N-SEL-ASYNC]. We have $\bigoplus_{i \in I} !l_i\langle S_i\rangle.T_i = T \not\lhd S$, with $\oplus \notin S$. Since T is a selection, $S \leqslant_{\mathsf{a}} T$ cannot hold by [SUB-END] nor [SUB-BRA]; moreover, it cannot hold by [SUB-SEL], since S cannot be a selection (otherwise, we would have the contradiction $\oplus \in S$); also [SUB-PERM-ASYNC] is ruled out: otherwise, we would have $S = \mathcal{A}[\bigoplus_{i \in I \cup J_n} !l_i\langle S_i^n\rangle.T_i^n]^{n \in N}$, and thus the contradiction $\oplus \in S$. Hence, we conclude $S \not\leqslant_{\mathsf{a}} T$;
- inductive case [N-CONT-ASYNC]. Then, we have:
  - $T = \bigoplus_{i \in I} !l_i\langle S_i\rangle.T_i$;
  - $S = \mathcal{A}[\bigoplus_{j \in J_n} !l_j^n\langle S_j^n\rangle.T_j^n]^{n \in N}$;
  - $\exists i_0 \in I, n_0 \in N, j_0 \in J_{n_0} : l_{j_0}^{n_0} = l_{i_0}$ and $T_{i_0} \not\lhd \mathcal{A}[T_{j_0}^n]^{n \in N}$.

  From the last item, by the induction hypothesis we have $T_{i_0} \not\leqslant_{\mathsf{a}} \mathcal{A}[T_{j_0}^n]^{n \in N}$. We observe that, since T is a selection, $T \leqslant_{\mathsf{a}} S$ could only possibly hold by rule [SUB-PERM-ASYNC], or [SUB-SEL] when $\mathcal{A}$ is just one hole. Since $T_{i_0} \not\leqslant_{\mathsf{a}} \mathcal{A}[T_{j_0}^n]^{n \in N}$, in both cases at least one of the coinductive premises of the candidate rule is not satisfied. Hence, we conclude $T \not\leqslant_{\mathsf{a}} S$.

Vice versa, the proof for $T \not\leqslant_{\mathsf{a}} S$ implies $T \not\vartriangleleft_{\mathsf{a}} S$ is similar to that for $T \not\leqslant_{\mathsf{s}} S$ implies $T \not\vartriangleleft_{\mathsf{s}} S$ (Lemma 3.3): we take a tentative derivation for $T \leqslant_{\mathsf{a}} S$ with a branch that fails after n steps, and by induction on n we construct a derivation of depth $n+1$ which concludes $T \not\vartriangleleft_{\mathsf{a}} S$. The only differences are the following:

- in the base case $n = 0$, we observe that if T and S cause an immediate derivation failure under $\leqslant_{\mathsf{s}}$, then they *also* cause an immediate failure under $\leqslant_{\mathsf{a}}$ — *except* when T is a selection and S is a branching. In this latter case, we must consider that rule [sub-perm-async] *might* allow for a further derivation step under $\leqslant_{\mathsf{a}}$; when this does *not* happen (i.e., the shapes of T and S do not match the conclusion of [sub-perm-async]), we construct a derivation of depth $1 = n+1$ which concludes $T \not\vartriangleleft_{\mathsf{a}} S$, by one of the axioms [n-label-async], [n-bra-async] or [n-sel-async];
- in the inductive case $n = m+1$, we must also consider the case in which the shapes of T, S match the conclusion of rule [sub-perm-async], but there is some coinductive premise $T' \leqslant_{\mathsf{a}} S'$ whose sub-derivation has a branch that fails after m steps. Then, by the induction hypothesis there exists a derivation of depth $m+1$ that concludes $T' \not\vartriangleleft_{\mathsf{a}} S'$; using this as a premise, by [n-exch-async] or [n-cont-async] we construct a derivation of depth $(m+1)+1 = n+1$ which concludes $T \not\vartriangleleft_{\mathsf{a}} S$. □

Before proving completeness of subtyping, we need two more intermediate results on the predicates $\& \in T$, $\& \notin T$, and $\oplus \in T$ (defined on pages 22 and 28).

Proposition 5.4 can be easily proved by noticing that dualisation turns branchings in the continuation paths of $\mathcal{T}(T)$ into selections in the continuation paths of $\mathcal{T}(\overline{T})$, and *vice versa*, as remarked on page 10.

Proposition 5.4. $\& \in T$ if and only if $\oplus \in \overline{T}$.

We define duality of asynchronous contexts as expected:

$$\overline{[\,]^n} = [\,]^n \qquad \overline{\&_{i\in I}?l_i(S_i).\mathcal{A}_i} = \bigoplus_{i\in I} !l_i\langle S_i\rangle.\overline{\mathcal{A}_i}$$

We use $\mathcal{B}$ to range over duals of asynchronous contexts. We extend asynchronous subtyping to the duals of asynchronous contexts in the obvious way:

$$[\text{sub-empty}]\quad [\,]^n \leqslant [\,]^n \qquad \frac{[\text{sub-dual-cont}]\quad \forall i \in I : S'_i \leqslant S_i \quad \mathcal{B}_i \leqslant \mathcal{B}'_i}{\bigoplus_{i\in I} !l_i\langle S_i\rangle.\mathcal{B}_i \leqslant \bigoplus_{i\in I\cup J} !l_i\langle S'_i\rangle.\mathcal{B}'_i}$$

The following lemma assures that, if there are continuation paths in the tree of a type T which does not contain branchings, then we can find a type S which is smaller than T (according to the synchronous subtyping, and then also according to the asynchronous one) such that all continuation paths in the tree of S do not contain branchings. Moreover T and S share similar structures.

Lemma 5.5. *If $\& \notin T$, then there is $S \leqslant_{\mathsf{s}} T$ such that the continuation paths of $\mathcal{T}(S)$ do not contain branchings. Moreover $T = \mathcal{B}[T_n]^{n\in N}$ and $S = \mathcal{B}'[T_n]^{n\in N'}$ with $\mathcal{B}' \leqslant \mathcal{B}$ and $N' \subseteq N$.*

Proof. If $\&$ does not occur in the continuation paths of $\mathcal{T}(T)$ we can choose $S = T$. Otherwise by definition $\mathcal{T}(T)$ contains some continuation paths with occurrences of $\&$ and other continuation paths without occurrences of $\&$. The continuation paths with occurrences of

& must contain nodes labelled by selections. We can then choose S as the session type whose tree is obtained by pruning top-down from selection nodes the continuation paths containing & and the exchanged sub-trees with the same label in $\mathcal{T}(T)$. We then get $S \leqslant_{\mathsf{s}} T$ through a derivation only composed by instances of rule [SUB-SEL]. This simple construction implies that $T = \mathcal{B}[T_n]^{n\in N}$ and $S = \mathcal{B}'[T_n]^{n\in N'}$ with $\mathcal{B}' \leqslant \mathcal{B}$ and $N' \subseteq N$. □

A last lemma connects tree representations of types and occurrences of channels as subjects in characteristic processes.

Lemma 5.6. *If the continuation paths of $\mathcal{T}(T)$ have no branchings, then $a \notin \varphi(\boldsymbol{P}(a, T))$.*

Proof. We first observe that, by Definition 5.1, the exchanged types of T do not influence whether a belongs to $\varphi(\boldsymbol{P}(a, T))$. Second, we naturally extend type trees to *open* types (with *closed* exchanged types) by letting $\mathcal{T}(\mathbf{t}) = \mathbf{t}$: this allows us to prove the statement by structural induction on T, neglecting its exchanged types. We show that if the continuation paths of $\mathcal{T}(T)$ do not contain branchings then $a \notin \delta(\boldsymbol{P}(a, T), \emptyset, \emptyset)$, considering only the most interesting cases:

- base case $T = \mathbf{t}$. Then, $\mathcal{T}(T)$ does not contain branchings and, by definition of δ (Table 12), we also have $a \notin \delta(X_{\mathbf{t}}\langle a\rangle, \emptyset, \emptyset) = \emptyset$;
- inductive case $T = \mu\mathbf{t}.T'$. We observe that:
  (1) the tree $\mathcal{T}(T)$ is obtained by recursively grafting the tree $\mathcal{T}(T')$ on its own leaf nodes marked with $\mathbf{t}$. Therefore, the continuation paths of $\mathcal{T}(T)$ do not contain branchings if and only if the continuation paths of $\mathcal{T}(T')$ do not contain branchings;
  (2) assume that $\mathcal{T}(T)$ does not contain branchings. From the previous item and by the induction hypothesis, we have $a \notin \delta\left(\boldsymbol{P}(a, T'), \emptyset, \emptyset\right)$;
  (3) by Definition 5.1, $\boldsymbol{P}(a, T) = \mathtt{def}\ X_{\mathbf{t}}(x) = \boldsymbol{P}(x, T')\ \mathtt{in}\ X_{\mathbf{t}}\langle a\rangle$. Therefore, we have $\delta(\boldsymbol{P}(a, T), \emptyset, \emptyset) = \delta\left(\boldsymbol{P}(a, T'),\, (X_{\mathbf{t}}(x) = \boldsymbol{P}(x, T'))\,,\, \emptyset\right)$, by definition of δ.

  Now, by contradiction, assume $a \in \delta(\boldsymbol{P}(a, T), \emptyset, \emptyset)$. By definition of δ, this means that a is yielded by a syntactic occurrence of either $u!l\langle a\rangle.P$ or $\sum_{i\in I} a?l_i(x_i).P_i$ in $\boldsymbol{P}(a, T')$. The former is impossible, because by Definition 5.1, the parameter a of $\boldsymbol{P}(a, T')$ is never sent as an exchanged channel. The latter, instead, by item 3 above implies:

$$a \;\in\; \delta\left(\boldsymbol{P}(a, T'),\, \left(X_{\mathbf{t}}(x) = \boldsymbol{P}(x, T')\right)\,,\, \emptyset\right)$$

  and thus $a \in \delta\left(\boldsymbol{P}(a, T'), \emptyset, \emptyset\right)$ — which contradicts item 2 above.

Finally, assume that the continuation paths of $\mathcal{T}(T)$ do not contain branchings: we have proved that $a \notin \delta(\boldsymbol{P}(a, T), \emptyset, \emptyset)$; by Definition 4.1, we conclude $a \notin \varphi(\boldsymbol{P}(a, T))$. □

Completeness can now be shown:

Theorem 5.7 (Completeness for asynchronous subtyping)**.** *The asynchronous subtyping relation $\leqslant_{\mathsf{a}}$ is complete for the asynchronous calculus.*

Proof. We prove that $T \not\leqslant_{\mathsf{a}} S$ implies that there are $T' \leqslant_{\mathsf{s}} T$ and $S' \leqslant_{\mathsf{s}} \overline{S}$, with either $T' = T$ or $S' = \overline{S}$, such that

$$(\nu ab)(\boldsymbol{P}(a, T') \mid \boldsymbol{P}(b, S') \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing) \rightarrow^*_{\mathsf{a}} \mathtt{error} \tag{5.1}$$

where $\boldsymbol{P}(a, T')$, $\boldsymbol{P}(b, S')$ are characteristic asynchronous processes. Note that $\boldsymbol{P}(a, T')$ and $\boldsymbol{P}(b, S')$ play respectively the rôles of P and Q in **Step 3** (page 12): in fact, we have $\boldsymbol{P}(a, T') \triangleright \{a : T\}$ and $\boldsymbol{P}(b, S') \triangleright \{b : \overline{S}\}$, by using rule [T-SUB] when $T' \neq T$ or $S' \neq \overline{S}$. The

proof is by induction on the derivation of $T \not\asymp_{\mathsf{a}} S$. In all inductive cases, we assume $T' \leqslant_{\mathsf{s}} T$ and $S' = \overline{S}$ as induction hypothesis; the proof assuming $T' = T$ and $S' \leqslant_{\mathsf{s}} \overline{S}$ is symmetric.

We do not consider the rules [N-LABEL-SEL], [N-EXCH-SEL] and [N-CONT-SEL], since they are particular cases of the rules [N-LABEL-ASYNC], [N-EXCH-ASYNC] and [N-CONT-ASYNC] when the asynchronous context is empty.

Case [N-END R]: $T = \mathtt{end}$ and $S \neq \mathtt{end}$.

$$(\nu ab)(\boldsymbol{P}(a,T) \mid \boldsymbol{P}(b,\overline{S}) \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing) = (\nu ab)(\mathbf{0} \mid \boldsymbol{P}(b,\overline{S}) \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing) \rightarrow_{\mathsf{a}} \mathtt{error}$$

by rules [ERR-IN-ASYNC] and [ERR-CONTEXT], since $a \notin \mathsf{fc}(\mathbf{0})$.

Case [N-END L]: $T \neq \mathtt{end}$ and $S = \mathtt{end}$. The proof is as in the previous case.

Case [N-BRASEL]: $T = \&_{i\in I} ?l_i(T'_i).T_i$ and $S = \bigoplus_{j\in J} !l'_j\langle S'_j\rangle.S_j$.

$$\begin{array}{l}(\nu ab)(\boldsymbol{P}(a,T) \mid \boldsymbol{P}(b,\overline{S}) \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing) = \\ (\nu ab)(\sum_{i\in I} \boldsymbol{P}_?(a,l_i,T'_i,T_i) \mid \sum_{j\in J} \boldsymbol{P}_?(b,l'_j,S'_j,\overline{S_j}) \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing) \rightarrow_{\mathsf{a}} \mathtt{error}\end{array}$$

by rules [ERR-IN-IN-ASYNC] and [ERR-CONTEXT].

Case [N-LABEL-BRA]: $T = \&_{i\in I} ?l_i(T'_i).T_i$, $S = \&_{j\in J} ?l'_j(S'_j).S_j$, and $\exists k \in J$ such that $\forall i \in I : l'_k \neq l_i$.

$$\begin{array}{l}(\nu ab)(\boldsymbol{P}(a,T) \mid \boldsymbol{P}(b,\overline{S}) \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing) \rightarrow_{\mathsf{a}} \\ (\nu ab)(\sum_{i\in I} \boldsymbol{P}_?(a,l_i,T'_i,T_i) \mid (\nu cd)(b!l'_k\langle c\rangle.\boldsymbol{P}(b,\overline{S_k}) \mid \boldsymbol{P}(d,\overline{S'_k}) \mid \\ \qquad\qquad dc \blacktriangleright \varnothing \mid cd \blacktriangleright \varnothing) \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing) \rightarrow_{\mathsf{a}} \\ C[\sum_{i\in I} \boldsymbol{P}_?(a,l_i,T'_i,T_i) \mid ba \blacktriangleright l'_k\langle c\rangle)]\end{array}$$

where $C = (\nu ab)(\nu cd)(\boldsymbol{P}(b,\overline{S_k}) \mid \boldsymbol{P}(d,\overline{S'_k}) \mid [\ ] \mid ab \blacktriangleright \varnothing \mid dc \blacktriangleright \varnothing \mid cd \blacktriangleright \varnothing)$.
By rule [ERR-MISM-ASYNC]

$$\textstyle\sum_{i\in I} \boldsymbol{P}_?(a,l_i,T'_i,T_i) \mid ba \blacktriangleright l'_k\langle c\rangle \rightarrow_{\mathsf{a}} \mathtt{error}$$

By rule [ERR-CONTEXT], we conclude

$$C[\textstyle\sum_{i\in I} \boldsymbol{P}_?(a,l_i,T'_i,T_i) \mid ba \blacktriangleright l'_k\langle c\rangle] \rightarrow_{\mathsf{a}} \mathtt{error}$$

Case [N-EXCH-BRA]: $T = \&_{i\in I} ?l_i(T'_i).T_i$, $S = \&_{j\in J} ?l'_j(S'_j).S_j$, and $\exists k \in I\ \exists k' \in J$ such that $l_k = l'_{k'}$ and $T'_k \not\asymp_{\mathsf{a}} S'_{k'}$. By induction there are $T^* \leqslant_{\mathsf{s}} T'_k$ and $S^* = \overline{S'_{k'}}$ such that

$$(\nu cd)(\boldsymbol{P}(c,T^*) \mid \boldsymbol{P}(d,S^*) \mid dc \blacktriangleright \varnothing \mid cd \blacktriangleright \varnothing) \rightarrow^*_{\mathsf{a}} \mathtt{error}$$

We can then choose

$$T' = ?l_k(T^*).T_k \ \&\ \&_{i\in I, i\neq k} ?l_i(T'_i).T_i \quad \text{and} \quad S' = !l_{k'}\langle\overline{S^*}\rangle.\overline{S_{k'}} \ \oplus\ \bigoplus_{j\in J, j\neq k'} !l'_j\langle S'_j\rangle.\overline{S_j}.$$

By definition $T' \leqslant_{\mathsf{s}} T$ and $S' = \overline{S}$. We get

$$\begin{array}{l}(\nu ab)(\boldsymbol{P}(a,T') \mid \boldsymbol{P}(b,S') \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing) \rightarrow_{\mathsf{a}} \\ (\nu ab)(\sum_{i\in I, i\neq k} \boldsymbol{P}_?(a,l_i,T'_i,T_i) + a?l_k(x).(\boldsymbol{P}(a,T_k) \mid \boldsymbol{P}(x,T^*))\mid \\ \qquad (\nu cd)(b!l_k\langle c\rangle.\boldsymbol{P}(b,\overline{S_{k'}}) \mid \boldsymbol{P}(d,S^*) \mid dc \blacktriangleright \varnothing \mid cd \blacktriangleright \varnothing) \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing) \rightarrow^*_{\mathsf{a}} \\ C[(\nu cd)(\boldsymbol{P}(c,T^*) \mid \boldsymbol{P}(d,S^*) \mid dc \blacktriangleright \varnothing \mid cd \blacktriangleright \varnothing)]\end{array}$$

where
$C[\,] = (\nu ab)(\boldsymbol{P}(a,T_k) \mid \boldsymbol{P}(b,\overline{S_{k'}}) \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing) \mid [\,]$. Then by rule [ERR-CONTEXT] we conclude

$$C[(\nu cd)(\boldsymbol{P}(c,T^*) \mid \boldsymbol{P}(d,S^*) \mid dc \blacktriangleright \varnothing \mid cd \blacktriangleright \varnothing)] \rightarrow^*_{\mathsf{a}} \mathtt{error}$$

Case [N-CONT-BRA]: $T = \&_{i\in I}?l_i(T'_i).T_i$, $S = \&_{j\in J}?l'_j(S'_j).S_j$, and $\exists k \in I\ \exists k' \in J$ such that $l_k = l'_{k'}$ and $T_k \not\downarrow_{\mathsf{a}} S_{k'}$. By induction there are $T^* \leqslant_{\mathsf{s}} T_k$ and $S^* = \overline{S_{k'}}$ such that

$$(\nu ab)(\boldsymbol{P}(a, T^*) \mid \boldsymbol{P}(b, S^*) \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing) \rightarrow^*_{\mathsf{a}} \texttt{error}$$

We can then choose

$$T' = ?l_k(T'_k).T^* \ \&\ \&_{i\in I, i\neq k}?l_i(T'_i).T_i \quad \text{and} \quad S' = !l_{k'}\langle S'_{k'}\rangle.S^* \ \oplus\ \bigoplus\nolimits_{j\in J, j\neq k'} !l'_j\langle S'_j\rangle.\overline{S_j}.$$

By definition $T' \leqslant_{\mathsf{s}} T$ and $S' = \overline{S}$. We get

$$\begin{array}{l}
(\nu ab)(\boldsymbol{P}(a, T') \mid \boldsymbol{P}(b, S') \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing) \rightarrow_{\mathsf{a}} \\
(\nu ab)(\sum_{i\in I, i\neq k} \boldsymbol{P}_?(a, l_i, T'_i, T_i) + a?l_k(x).(\boldsymbol{P}(a, T^*) \mid \boldsymbol{P}(x, T'_k))| \\
\qquad (\nu cd)(b!l_k\langle c\rangle.\boldsymbol{P}(b, S^*) \mid \boldsymbol{P}(d, \overline{S'_{k'}}) \mid dc \blacktriangleright \varnothing \mid cd \blacktriangleright \varnothing) \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing) \rightarrow_{\mathsf{a}} \\
(\nu ab)(\nu cd)(\sum_{i\in I, i\neq k} \boldsymbol{P}_?(a, l_i, T'_i, T_i) + a?l_k(x).(\boldsymbol{P}(a, T^*) \mid \boldsymbol{P}(x, T'_k))| \\
\qquad\qquad \boldsymbol{P}(b, S^*) \mid \boldsymbol{P}(d, \overline{S'_{k'}}) \mid ba \blacktriangleright l_k\langle c\rangle \mid ab \blacktriangleright \varnothing \mid dc \blacktriangleright \varnothing \mid cd \blacktriangleright \varnothing) \rightarrow_{\mathsf{a}} \\
C[(\nu ab)(\boldsymbol{P}(a, T^*) \mid \boldsymbol{P}(b, S^*) \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing)]
\end{array}$$

where

$$C[\ ] = (\nu cd)(\boldsymbol{P}(c, T'_k) \mid \boldsymbol{P}(d, \overline{S'_{k'}}) \mid dc \blacktriangleright \varnothing \mid cd \blacktriangleright \varnothing) \mid [\ ]$$

Then by rule [ERR-CONTEXT] we conclude

$$C[(\nu ab)(\boldsymbol{P}(a, T^*) \mid \boldsymbol{P}(b, S^*) \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing)] \rightarrow^*_{\mathsf{a}} \texttt{error}$$

Case [N-LABEL-ASYNC]: $T = \bigoplus_{i\in I} !l_i\langle T'_i\rangle.T_i$, $S = \mathcal{A}[\bigoplus_{j\in J_n} !{l'_j}^n\langle {S'_j}^n\rangle.S^n_j]^{n\in N}$ and there are $i_0 \in I, n_0 \in N$ such that $\forall j \in J_{n_0}$ we get ${l'_j}^{n_0} \neq l_{i_0}$. We show by induction on $\mathcal{A}$ that $T \not\downarrow_{\mathsf{a}} S$ implies

$$(\nu ab)(\boldsymbol{P}(a, T) \mid \boldsymbol{P}(b, \overline{S}) \mid ba \blacktriangleright h \mid ab \blacktriangleright \varnothing) \rightarrow^*_{\mathsf{a}} \texttt{error}$$

for an arbitrary queue h.

(1) If $\mathcal{A} = [\ ]$, then $S = \bigoplus_{j\in J} !l'_j\langle S'_j\rangle.S_j$, with $J = J_{n_0}$, and $\overline{S} = \&_{j\in J}?l'_j(S'_j).\overline{S_j}$. We get

$$\begin{array}{l}
(\nu ab)(\boldsymbol{P}(a, T) \mid \boldsymbol{P}(b, \overline{S}) \mid ba \blacktriangleright h \mid ab \blacktriangleright \varnothing) \rightarrow_{\mathsf{a}} \\
(\nu ab)(\boldsymbol{P}_!(a, l_{i_0}, T'_{i_0}, T_{i_0}) \mid \sum_{j\in J} \boldsymbol{P}_?(b, l'_j, S'_j, \overline{S_j}) \mid ba \blacktriangleright h \mid ab \blacktriangleright \varnothing) \rightarrow_{\mathsf{a}} \\
C[\sum_{j\in J} b?l'_j(x).(\boldsymbol{P}(b, \overline{S_j}) \mid \boldsymbol{P}(x, S'_j)) \mid ab \blacktriangleright l_{i_0}\langle c\rangle]
\end{array}$$

where

$$C[\ ] = (\nu ab)(\nu cd)(\boldsymbol{P}(a, T_{i_0}) \mid \boldsymbol{P}(d, \overline{T'_{i_0}}) \mid [\ ] \mid ba \blacktriangleright h \mid dc \blacktriangleright \varnothing \mid cd \blacktriangleright \varnothing)$$

By rule [ERR-MISM-ASYNC]

$$\sum_{j\in J} b?l'_j(x).\boldsymbol{P}(b, \overline{S_j}) \mid ab \blacktriangleright l_{i_0}\langle c\rangle \rightarrow_{\mathsf{a}} \texttt{error}$$

then by rule [ERR-CONTEXT], we conclude

$$C[\sum_{j\in J} b?l'_j(x).\boldsymbol{P}(b, \overline{S_j}) \mid ab \blacktriangleright l_{i_0}\langle c\rangle] \rightarrow_{\mathsf{a}} \texttt{error}$$

(2) If $\& \in \mathcal{A}$, let $\mathcal{A} = \&_{k\in K}?l^*_k(S^*_k).\mathcal{A}_k[\ ]^{n\in N_k}$, where $\bigcup_{k\in K} N_k = N$. Then

$$\overline{S} \ = \ \bigoplus_{k\in K} !l^*_k\langle S^*_k\rangle.\overline{\mathcal{A}_k}[\&_{j\in J_n}?{l'_j}^n({S'_j}^n).\overline{S^n_j}]^{n\in N_k}$$

Let $k_0 \in K$ be such that $n_0 \in N_{k_0}$ and

$$V = \overline{\mathcal{A}_{k_0}}[\&_{j\in J_n}?{l'_j}^n({S'_j}^n).\overline{S^n_j}]^{n\in N_{k_0}}$$

We get

$(\nu ab)(\boldsymbol{P}(a,T) \mid \boldsymbol{P}(b,\overline{S}) \mid ba \blacktriangleright h \mid ab \blacktriangleright \varnothing) \to_{\mathsf{a}}$
$(\nu ab)(\boldsymbol{P}(a,T) \mid (\nu cd)(b!l^*_{k_0}\langle c\rangle.\boldsymbol{P}(b,V) \mid \boldsymbol{P}(d,\overline{S^*_{k_0}}) \mid dc \blacktriangleright \varnothing \mid cd \blacktriangleright \varnothing) \mid ba \blacktriangleright h \mid ab \blacktriangleright \varnothing) \to_{\mathsf{a}}$
$C[(\nu ab)(\boldsymbol{P}(a,T) \mid \boldsymbol{P}(b,V) \mid ba \blacktriangleright h \cdot l^*_{k_0}\langle c\rangle \mid ab \blacktriangleright \varnothing)]$

where

$$C = (\nu cd)(\boldsymbol{P}(d,\overline{S^*_{k_0}}) \mid [\ ] \mid dc \blacktriangleright \varnothing \mid cd \blacktriangleright \varnothing)$$

By induction

$$(\nu ab)(\boldsymbol{P}(a,T) \mid \boldsymbol{P}(b,V) \mid ba \blacktriangleright h \cdot l^*_{k_0}\langle c\rangle \mid ab \blacktriangleright \varnothing) \to^*_{\mathsf{a}} \texttt{error}$$

then by rule [ERR-CONTEXT], we conclude

$$C[(\nu ab)(\boldsymbol{P}(a,T) \mid \boldsymbol{P}(b,V) \mid ba \blacktriangleright h \cdot l^*_{k_0}\langle c\rangle \mid ab \blacktriangleright \varnothing)] \to^*_{\mathsf{a}} \texttt{error}$$

Case [N-EXCH-ASYNC]: $T = \bigoplus_{i\in I} !l_i\langle T'_i\rangle.T_i$, $S = \mathcal{A}[\bigoplus_{j\in J_n} !l'^{\,n}_j\langle S'^{\,n}_j\rangle.S^n_j]^{n\in N}$, and there are $i_0 \in I, n_0 \in N, j_0 \in J_{n_0}$ such that $l'^{n_0}_{j_0} = l_{i_0}$ and $S'^{n_0}_{j_0} \not\dashv_{\mathsf{a}} T'_{i_0}$. We show by induction on $\mathcal{A}$ that $T \not\dashv_{\mathsf{a}} S$ implies

$$(\nu ab)(\boldsymbol{P}(a,T') \mid \boldsymbol{P}(b,S') \mid ba \blacktriangleright h \mid ab \blacktriangleright \varnothing) \to^*_{\mathsf{a}} \texttt{error}$$

for some $T' = T$, $S' \leqslant_{\mathsf{s}} \overline{S}$ and an arbitrary queue h.

(1) If $\mathcal{A} = [\ ]$, then $S = \bigoplus_{j\in J} !l'_j\langle S'_j\rangle.S_j$, where $J = J_{n_0}$, and $\overline{S} = \&_{j\in J} ?l'_j(S'_j).\overline{S_j}$ and $S'_{j_0} \not\dashv_{\mathsf{a}} T'_{i_0}$. By induction there are $T^* = \overline{T'_{i_0}}$ and $S^* \leqslant_{\mathsf{s}} S'_{j_0}$ such that

$$(\nu cd)(\boldsymbol{P}(c,S^*) \mid \boldsymbol{P}(d,T^*) \mid dc \blacktriangleright \varnothing \mid cd \blacktriangleright \varnothing) \to^*_{\mathsf{a}} \texttt{error}$$

We can choose $T' = !l_{i_0}\langle\overline{T^*}\rangle.T_{i_0} \oplus \bigoplus_{i\in I, i\neq i_0} !l_i\langle T'_i\rangle.T_i$ (therefore, $T' = T$) and $S' = ?l_{i_0}(S^*).\overline{S_{j_0}} \,\&\, \&_{j\in J, j\neq j_0} ?l_j(S'_j).\overline{S_j}$ (therefore, $S' \leqslant_{\mathsf{s}} \overline{S}$).We get

$(\nu ab)(\boldsymbol{P}(a,T') \mid \boldsymbol{P}(b,S') \mid ba \blacktriangleright h \mid ab \blacktriangleright \varnothing) \to_{\mathsf{a}}$
$(\nu ab)(\nu cd)(\boldsymbol{P}(a,T_{i_0}) \mid b?l_{i_0}(x).(\boldsymbol{P}(b,\overline{S_j}) \mid \boldsymbol{P}(x,S^*)) + \sum_{j\in J, j\neq j_0} b?l'_j(x).(\boldsymbol{P}(b,\overline{S_j}) \mid \boldsymbol{P}(x,S'_j)) \mid$
$\boldsymbol{P}(d,T^*) \mid ba \blacktriangleright h \mid ab \blacktriangleright l_{i_0}\langle c\rangle \mid dc \blacktriangleright \varnothing \mid cd \blacktriangleright \varnothing) \to_{\mathsf{a}}$
$C[(\nu cd)(\boldsymbol{P}(c,S^*) \mid \boldsymbol{P}(d,T^*) \mid dc \blacktriangleright \varnothing \mid cd \blacktriangleright \varnothing)]$

where $C = (\nu ab)(\boldsymbol{P}(a,T_{i_0}) \mid \boldsymbol{P}(b,\overline{S_{j_0}}) \mid ba \blacktriangleright h \mid ab \blacktriangleright \varnothing) \mid [\ ]$. Then by [ERR-CONTEXT], we conclude

$$C[(\nu cd)(\boldsymbol{P}(c,S^*) \mid \boldsymbol{P}(d,T^*) \mid dc \blacktriangleright \varnothing \mid cd \blacktriangleright \varnothing)] \to^*_{\mathsf{a}} \texttt{error}$$

(2) If $\& \in \mathcal{A}$, then the proof is as in case (2) of rule [N-LABEL-ASYNC].

Case [N-CONT-ASYNC]: $T = \bigoplus_{i\in I} !l_i\langle T'_i\rangle.T_i$, $S = \mathcal{A}[\bigoplus_{j\in J_n} !l'^{\,n}_j\langle S'^{\,n}_j\rangle.S^n_j]^{n\in N}$, and for all $i \in I, n \in N:\ \exists j_{i,n} \in J_n$ such that $l'^{\,n}_{j_{i,n}} = l_i$ and $\exists i_0 \in I : T_{i_0} \not\dashv \mathcal{A}[S^n_{j_{i_0,n}}]^{n\in N}$. By induction, there exist $T^* \leqslant_{\mathsf{s}} T_{i_0}$ and $S^* = \overline{\mathcal{A}[S^n_{j_{i_0,n}}]^{n\in N}}$ such that:

$$(\nu ab)(\boldsymbol{P}(a,T^*) \mid \boldsymbol{P}(b,S^*) \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing) \to^*_{\mathsf{a}} \texttt{error} \tag{5.2}$$

By Lemma 5.5 $S^* = \mathcal{B}[S^n_*]^{n\in N'}$ with $\mathcal{B} \leqslant \overline{\mathcal{A}}$, $N' \subseteq N$ and $S^n_* = \overline{S^n_{j_{i_0,n}}}$. We observe that (5.2) implies that in $\mathcal{B}$ there exists a continuation path of m outputs $!l^\flat_1\langle S^\flat_1\rangle, \ldots, !l^\flat_m\langle S^\flat_m\rangle$

reaching a k-indexed hole, such that if $\boldsymbol{P}(b, S^*)$ fires the m outputs along such a path, we get[1]:

$$(\nu ab)(\boldsymbol{P}(a, T^*) \mid \boldsymbol{P}(b, S^*) \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing) \overbrace{\rightarrow_{\mathsf{a}} \cdots \rightarrow_{\mathsf{a}}}^{m \text{ times}} \tag{5.3}$$

$$C\left[(\nu ab)\left(\boldsymbol{P}(a, T^*) \mid \boldsymbol{P}(b, S_*^k) \mid ba \blacktriangleright l_1^\flat\langle c_1^\flat\rangle \cdot \ldots \cdot l_m^\flat\langle c_m^\flat\rangle \mid ab \blacktriangleright \varnothing\right)\right] \rightarrow_{\mathsf{a}}^* \texttt{error} \tag{5.4}$$

where $C[\ ]$ contains the restrictions and the characteristic processes for the exchanged channels $c_1^\flat, \ldots, c_m^\flat$.

Now, we can choose $T' = !l_{i_0}\langle T'_{i_0}\rangle.T^* \oplus \bigoplus_{i\in I, i\neq i_0} !l_i\langle T'_i\rangle.T_i$ (therefore, $T' \leqslant_{\mathsf{s}} T$) and $S' = \mathcal{B}[?l_{i_0}(S'^n_{j_{i_0,n}}).S_*^n \ \&\ \&_{j\in J_n, j\neq j_{i_0,n}} ?{l'_j}^n({S'_j}^n).\overline{S_j^n}^{n\in N'}$ (therefore, $S' = \overline{S}$). We have:

$$(\nu ab)(\boldsymbol{P}(a, T') \mid \boldsymbol{P}(b, S') \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing) \rightarrow_{\mathsf{a}} \tag{5.5}$$

$$\begin{gathered} C'\left[(\nu ab)(\boldsymbol{P}(a, T^*) \mid \boldsymbol{P}(b, S') \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright l_{i_0}\langle c_{i_0}\rangle)\right] \overbrace{\rightarrow_{\mathsf{a}} \cdots \rightarrow_{\mathsf{a}}}^{m \text{ times}} \\ C'\Big[C\Big[(\nu ab)\Big(\boldsymbol{P}(a, T^*) \mid \boldsymbol{P}(b, ?l_{i_0}(S'^k_{j_{i_0,k}}).S_*^k \ \&\ \&_{j\in J_k, j\neq j_{i_0,k}} ?{l'_j}^k({S'_j}^k).\overline{S_j^k}) \\ \mid ba \blacktriangleright !l_1^\flat\langle c_1^\flat\rangle \cdot \ldots \cdot !l_m^\flat\langle c_m^\flat\rangle \mid ab \blacktriangleright l_{i_0}\langle c_{i_0}\rangle\Big)\Big]\Big] \rightarrow_{\mathsf{a}} \\ C''\left[C\left[(\nu ab)\left(\boldsymbol{P}(a, T^*) \mid \boldsymbol{P}(b, S_*^k) \mid ba \blacktriangleright l_1^\flat c_1^\flat \cdot \ldots \cdot l_m^\flat c_m^\flat \mid ab \blacktriangleright \varnothing\right)\right]\right] \end{gathered} \tag{5.6}$$

where $C'[\ ]$ and $C''[\ ]$ contain the restrictions and the characteristic processes for the exchanged channel c_{i_0} and its co-channel. The reductions from (5.3) and (5.5) perform the same communications, except for the enqueuing/dequeuing of $l_{i_0}\langle c_{i_0}\rangle$ on ab. Moreover, the reached configurations (5.4) and (5.6) coincide, except for the surrounding context $C''[\ ]$. Thus, all errors reachable from (5.4) are also reachable from (5.6), by [ERR-CONTEXT]. We conclude $(\nu ab)(\boldsymbol{P}(a, T') \mid \boldsymbol{P}(b, S') \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing) \rightarrow_{\mathsf{a}}^* \texttt{error}$.

Case [N-BRA-ASYNC]: $\& \notin T$ and $S = \&_{i\in I} ?l_i(S_i).T_i$. Then $\overline{S} = \bigoplus_{i\in I} !l_i\langle S_i\rangle.\overline{T_i}$. By Lemma 5.5 there is $T' \leqslant_{\mathsf{s}} T$ such that the continuation paths of $\mathcal{T}(T')$ do not contain branchings.

$$\begin{gathered} (\nu ab)(\boldsymbol{P}(a, T') \mid \boldsymbol{P}(b, \overline{S}) \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing) \rightarrow_{\mathsf{a}} \\ (\nu ab)(\boldsymbol{P}(a, T') \mid \boldsymbol{P}_!(b, l_{i_0}, S_{i_0}, \overline{T_{i_0}}) \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing) \rightarrow_{\mathsf{a}} \\ C[(\nu ab)(\boldsymbol{P}(a, T') \mid \boldsymbol{P}(b, \overline{T_{i_0}}) \mid ba \blacktriangleright l_{i_0}\langle c\rangle \mid ab \blacktriangleright \varnothing)] \end{gathered}$$

where

$$C[\ ] = (\nu cd)(\boldsymbol{P}(d, \overline{S_{i_0}}) \mid [\ ] \mid dc \blacktriangleright \varnothing \mid cd \blacktriangleright \varnothing)$$

By Lemma 5.6 $a \notin \varphi(\boldsymbol{P}(a, T') \mid \boldsymbol{P}(b, \overline{T_{i_0}}) \mid ab \blacktriangleright \varnothing)$, since the continuation paths of $\mathcal{T}(T')$ do not contain branchings, and

$$\mathsf{fpv}(\boldsymbol{P}(a, T') \mid \boldsymbol{P}(b, \overline{T_{i_0}}) \mid ab \blacktriangleright \varnothing) = \emptyset$$

so by [ERR-ORPH-MESS-ASYNC] we get

$$(\nu ab)(\boldsymbol{P}(a, T') \mid \boldsymbol{P}(b, \overline{T_{i_0}}) \mid ba \blacktriangleright l_{i_0}\langle c\rangle \mid ab \blacktriangleright \varnothing) \rightarrow_{\mathsf{a}} \texttt{error}$$

Then by [ERR-CONTEXT], we conclude

$$C[(\nu ab)(\boldsymbol{P}(a, T') \mid \boldsymbol{P}(b, \overline{T_{i_0}}) \mid ba \blacktriangleright l_{i_0}\langle c\rangle \mid ab \blacktriangleright \varnothing)] \rightarrow_{\mathsf{a}}{}^* \texttt{error}$$

[1]Note that, if there are errors in the exchanged types, an `error` transition might be enabled *before* the whole continuation path is fired; in this case, by Proposition 4.4, it will remain enabled until (5.4) is reached.

Case [N-SEL-ASYNC]: $T = \bigoplus_{i\in I} !l_i\langle S_i\rangle.T_i$ and $\oplus \notin S$. By Proposition 5.4, we have $\& \notin \overline{S}$. By Lemma 5.5 there is $S' \leqslant_{\mathsf{s}} \overline{S}$ such that the continuation paths of $\mathcal{T}(S')$ do not contain branchings.

$$(\nu ab)(\boldsymbol{P}(a,T) \mid \boldsymbol{P}(b,S') \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing) \rightarrow_{\mathsf{a}}$$
$$(\nu ab)(\boldsymbol{P}(a,l_{i_0},S_{i_0},T_{i_0}) \mid \boldsymbol{P}_!(b,S') \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing) \rightarrow_{\mathsf{a}}$$
$$C[(\nu ab)(\boldsymbol{P}(a,T_{i_0}) \mid \boldsymbol{P}(b,S') \mid ba \blacktriangleright l_{i_0}\langle c\rangle \mid ab \blacktriangleright \varnothing)]$$

where $C[\,] = (\nu cd)(\boldsymbol{P}(d,\overline{S_{i_0}}) \mid [\,] \mid dc \blacktriangleright \varnothing \mid cd \blacktriangleright \varnothing)$.
By Lemma 5.6 $b \notin \varphi(\boldsymbol{P}(a,T_{i_0}) \mid \boldsymbol{P}(b,S') \mid ab \blacktriangleright \varnothing)$, since the continuation paths of $\mathcal{T}(S')$ do not contain branchings, and

$$\mathsf{fpv}(\boldsymbol{P}(a,T_{i_0}) \mid \boldsymbol{P}(b,S') \mid ab \blacktriangleright \varnothing) = \emptyset$$

so by [ERR-ORPH-MESS-ASYNC] we get

$$(\nu ab)(\boldsymbol{P}(a,T_{i_0}) \mid \boldsymbol{P}(b,S') \mid ba \blacktriangleright l_{i_0}\langle c\rangle \mid ab \blacktriangleright \varnothing) \rightarrow_{\mathsf{a}} \texttt{error}$$

Then by [ERR-CONTEXT], we conclude

$$C[(\nu ab)(\boldsymbol{P}(a,T_{i_0}) \mid \boldsymbol{P}(b,S') \mid ba \blacktriangleright l_{i_0}\langle c\rangle \mid ab \blacktriangleright \varnothing)] \rightarrow_{\mathsf{a}}{}^{*} \texttt{error}$$

Summing up, we proved that $T \not\leqslant_{\mathsf{a}} S$ implies that there are $T' \leqslant_{\mathsf{s}} T$ and $S' \leqslant_{\mathsf{s}} \overline{S}$, with either $T' = T$ or $S' = \overline{S}$, such that

$$(\nu ab)(\boldsymbol{P}(a,T') \mid \boldsymbol{P}(b,S') \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing) \rightarrow_{\mathsf{a}}^{*} \texttt{error}$$

Hence, the subtyping relation $\leqslant_{\mathsf{a}}$ is complete for the asynchronous calculus, according to Definition 1.1. □

We end this section with two examples showing why, in the proof for Theorem 5.7, for rules [N-BRA-ASYNC] and [N-SEL-ASYNC] we need to build characteristic processes of *subtypes* of the current types T and $\overline{S}$. By highlighting these two cases, we will then discuss the existence of other sound asynchronous subtypings and a further result on $\leqslant_{\mathsf{a}}$ (Theorem 5.10).

Example 5.8. *If we take:*

$$T = \mu\mathbf{t}.\big(!l_1\langle S_1\rangle.\mathbf{t} \oplus !l_2\langle S_2\rangle.?l_3(S_3).\texttt{end}\big) \qquad \text{and} \qquad S = ?l_3(S_3).\big(\mu\mathbf{t}.!l_1\langle S_1\rangle.\mathbf{t} \oplus !l_2\langle S_2\rangle.\texttt{end}\big)$$

then $T \not\leqslant_{\mathsf{a}} S$ *(by rule [*N-BRA-ASYNC*]), but* $(\nu ab)(\boldsymbol{P}(a,T) \mid \boldsymbol{P}(b,\overline{S}) \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing)$ *does* not *reduce to* `error`. *In fact, the process* $\boldsymbol{P}(a,T)$ *either sends an* l_1*-labelled message and becomes* $\boldsymbol{P}(a,T)$ *again, or it sends an* l_2*-labelled message, receives an* l_3*-labelled message and stops. The process* $\boldsymbol{P}(b,\overline{S})$ *sends one* l_3*-labelled message, and then can receive either an* l_1*-labelled message and become* $\boldsymbol{P}(b,\overline{S})$ *again, or an* l_2*-labelled message and stop. Hence, the* l_3*-labelled message can always be potentially dequeued from* ba *(after* $\boldsymbol{P}(a,T)$ *chooses to output* l_2*), and this ensures that [*ERR-ORPH-MESS-ASYNC*] never holds.*

Instead taking $T' = \mu\mathbf{t}.!l_1\langle S_1\rangle.\mathbf{t}$ *we get* $T' \leqslant_{\mathsf{a}} T$, *and*

$$(\nu ab)\big(\boldsymbol{P}(a,T') \mid \boldsymbol{P}(b,\overline{S}) \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing\big) \;\rightarrow_{\mathsf{a}}^{*}\; \texttt{error}$$

since $a \notin \varphi(\boldsymbol{P}(a,T'))$, *i.e.* $\boldsymbol{P}(a,T')$ *cannot read the* l_3*-message sent by* $\boldsymbol{P}(b,\overline{S})$.

Example 5.9. *This example is the "dual" of Example 5.8. If we take:*

$$T = \mu\mathbf{t}.!l_1\langle S_1\rangle.\big(?l_2(S_2).\mathbf{t} \;\&\; ?l_3(S_3).\mathbf{t}\big) \qquad \text{and} \qquad S = \mu\mathbf{t}.\big(?l_2(S_2).\mathbf{t} \;\&\; ?l_3(S_3).!l_1\langle S_1\rangle.\mathbf{t}\big)$$

then $T \not\leqslant_{\mathsf{a}} S$ *(by rule [*N-SEL-ASYNC*]), but* $(\nu ab)(\boldsymbol{P}(a,T) \mid \boldsymbol{P}(b,\overline{S}) \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing)$ *does* not *reduce to* `error`. *Instead taking* $S' = \mu\mathbf{t}.?l_2(S_2).\mathbf{t}$ *we get* $\overline{S'} \leqslant_{\mathsf{a}} \overline{S}$, *and*

$$(\nu ab)\big(\boldsymbol{P}(a,T) \mid \boldsymbol{P}(b,\overline{S'}) \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing\big) \;\rightarrow_{\mathsf{a}}^{*}\; \texttt{error}$$

Examples 5.8 and 5.9 show that there exist sound asynchronous subtyping relations that are *not* sub-relations of $\leqslant_{\mathsf{a}}$. For instance, take T and S from Example 5.8, and let $\sqsubseteq$ be the smallest reflexive relation between session types such that $T \sqsubseteq S$. We can verify that $\sqsubseteq$ is a sound subtyping, by Definition 1.1: if we take any $T' \sqsubseteq S'$ and we compose two processes typed by T' and $\overline{S'}$, they will not reduce to error. Note, however, that $\sqsubseteq$ is *not* a sub-relation of $\leqslant_{\mathsf{a}}$, because $T \sqsubseteq S$ but $T \not\leqslant_{\mathsf{a}} S$: this is unlike the synchronous calculus, where *all* sound subtypings are sub-relations of $\leqslant_{\mathsf{s}}$ (see proof of Corollary 3.5). However, $\leqslant_{\mathsf{a}}$ has an important property: if we only consider the asynchronous subtypings that extend $\leqslant_{\mathsf{s}}$, then $\leqslant_{\mathsf{a}}$ is the unique precise one.

Theorem 5.10. *$\leqslant_{\mathsf{a}}$ is the unique precise subtyping for the asynchronous calculus that extends $\leqslant_{\mathsf{s}}$.*

Proof. Take a reflexive and transitive relation $\sqsubseteq_{\mathsf{a}}$ such that $\leqslant_{\mathsf{s}} \subseteq \sqsubseteq_{\mathsf{a}} \not\subseteq \leqslant_{\mathsf{a}}$ — i.e., there exist T, S such that $T \sqsubseteq_{\mathsf{a}} S$ but $T \not\leqslant_{\mathsf{a}} S$. We prove that $\sqsubseteq_{\mathsf{a}}$ is an unsound subtyping, by showing that there exist $T' \sqsubseteq_{\mathsf{a}} T$ and $S' \sqsubseteq_{\mathsf{a}} \overline{S}$ such that

$$(\nu ab)(\boldsymbol{P}(a, T') \mid \boldsymbol{P}(b, S') \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing) \rightarrow_{\mathsf{a}}^{*} \mathtt{error}$$

Since (by Lemma 5.3) $T \not\leqslant_{\mathsf{a}} S$ implies $T \not\lhd_{\mathsf{a}} S$, we proceed by induction on the derivation of $T \not\lhd_{\mathsf{a}} S$, similarly to the proof of Theorem 5.7.

From such a proof, we can see that in all cases we can get the error reduction above for some T', S' such that either $T' \leqslant_{\mathsf{s}} T$ and $S' = \overline{S}$, or $T' = T$ and $S' \leqslant_{\mathsf{s}} \overline{S}$. In the first case, we also have $T' \sqsubseteq_{\mathsf{a}} T$ and $S' \sqsubseteq_{\mathsf{a}} \overline{S}$ (because $\leqslant_{\mathsf{s}} \subseteq \sqsubseteq_{\mathsf{a}}$): hence, we conclude that $\sqsubseteq_{\mathsf{a}}$ is an unsound subtyping according to Definition 1.1. The proof for the second case ($T' = T$ and $S' \leqslant_{\mathsf{s}} \overline{S}$) is dual.

We conclude that if $\sqsubseteq_{\mathsf{a}}$ is sound and $\leqslant_{\mathsf{s}} \subseteq \sqsubseteq_{\mathsf{a}}$, then $\sqsubseteq_{\mathsf{a}} \subseteq \leqslant_{\mathsf{a}}$; hence, among all subtypings that extend $\leqslant_{\mathsf{s}}$, the subtyping $\leqslant_{\mathsf{a}}$ is the largest sound one for the asynchronous calculus, and therefore the unique precise one. □

6. Extensions

In the original calculus [36] sessions are initialised using shared channels by request/accept prefixes and also expressions (including shared channels) can be communicated. In this section we show that preciseness is preserved when we augment the calculus, the types (adding sorts, following Honda *et al.* [36]) and the subtyping (following Demangeon and Honda [15]), both for the synchronous and for the asynchronous cases. The most challenging issue in this extension is the definition of characteristic processes. For the communication of expressions, taking inspiration from Ligatti *et al.* [44], we add expression constructors distinguishing values of different sorts. To correctly deal with communication of shared channels the characteristic processes must contain both accepts and requests. If a shared channel carries a linear channel of type S, an accept can be typed with a linear channel of type $T \leqslant S$, while a request can be typed with a linear channel of type $T \leqslant \overline{S}$, see Table 22. Since $T \leqslant \overline{S}$ is equivalent to $S \leqslant \overline{T}$, typing the parallel composition of accept and request is invariant, as defined by Demangeon and Honda [15].

$P ::=$	**Process**				
	$\vdots$	from Table 1	$\mid$	$u!l\langle \mathsf{e} \rangle.P$	(expression output)
$\mid$	$u(x).P$	(session accept)	$\mid$	$\overline{u}(x).P$	(session request)
$\mid$	$(\nu \mathsf{s})P$	(shared channel restriction)	$\mid$	$\mathtt{if}\ \mathsf{e}\ \mathtt{then}\ P\ \mathtt{else}\ P$	(conditional)
$\mid$	$X\langle \tilde{\mathsf{e}} \tilde{u} \rangle$	(variable with expressions)			

$u ::=$	**Identifiers**	
	$\vdots$	from Table 1
$\mid$	v	(value)

$\mathsf{v} ::=$	**Value**			$\mathsf{e} ::=$	**Expression**		
	s	(shared channel)			v	$\mid$ x	
$\mid$	$\mathtt{true}$	$\mid$ $\mathtt{false}$		$\mid$	$\neg \mathsf{e}$	$\mid$ $\mathtt{succ}(\mathsf{e})$	
$\mid$	0	$\mid$ 1		$\mid$	$\mathtt{neg}(\mathsf{e})$	$\mid$ $\mathsf{e} > \mathsf{e}$	
$\mid$	-1	$\mid$ $\ldots$					

Table 17: Syntax of extended synchronous processes.

$$\neg\mathtt{true} \downarrow \mathtt{false} \qquad \neg\mathtt{false} \downarrow \mathtt{true} \qquad \mathtt{succ}(\mathsf{n}) \downarrow (\mathsf{n}+1) \qquad \mathtt{neg}(\mathsf{i}) \downarrow (-\mathsf{i})$$

$$(\mathsf{i}_1 > \mathsf{i}_2) \downarrow \begin{cases} \mathtt{true} & \text{if } \mathsf{i}_1 > \mathsf{i}_2, \\ \mathtt{false} & \text{otherwise} \end{cases} \qquad \frac{\mathsf{e} \downarrow \mathsf{v} \quad \mathcal{E}(\mathsf{v}) \downarrow \mathsf{v}'}{\mathcal{E}(\mathsf{e}) \downarrow \mathsf{v}'} \qquad \mathsf{v} \downarrow \mathsf{v}$$

Table 18: Expression evaluation.

6.1. **Synchronous Communication.**

Syntax and operational semantics. Table 17 shows the extended synchronous session calculus obtained by adding session initialisations and communications of expressions (including shared channels) to the synchronous calculus of Table 1. We also add conditionals in order to get evaluation of expressions in reducing characteristic processes.

The value v of expression e (notation $\mathsf{e} \downarrow \mathsf{v}$) is computed according to the rules of Table 18. We use n to range over natural and i to range over integer numbers. Evaluation contexts $\mathcal{E}$ for expressions are defined by:

$$\mathcal{E} ::= [\,] \mid \neg(\mathcal{E}) \mid \mathtt{neg}(\mathcal{E}) \mid \mathtt{succ}(\mathcal{E}) \mid \mathcal{E} > \mathsf{e} \mid \mathsf{v} > \mathcal{E}$$

An expression e is *stuck* (notation $\mathsf{e} \not\downarrow$) if it does not evaluate to a value according to the rules of Table 18. Note that `succ` reduces only if the argument is a natural number. We use $\tilde{\mathsf{e}} \downarrow \tilde{\mathsf{v}}$ and $\tilde{\mathsf{e}} \not\downarrow$ with the obvious meanings.

We extend the structural congruence of synchronous processes (Table 2) and the evaluation contexts (see page 6) in the obvious way. To reduce extended processes we add the rules of Table 19 to the rules of Table 3. Table 19 takes into account session initialisations, communication of expressions, conditionals and expressions in definitions.

A process containing a stuck expression reduces to `error`, as well as a process with a value in a channel position, or with a shared channel in a linear channel position, or vice versa. This is prescribed by the rules of Table 20, which are added to the rules of Table 4.

$$[\text{R-INIT-SYNC}]\quad \frac{a, b \text{ fresh}}{\mathsf{s}(x).P \mid \overline{\mathsf{s}}(y).Q \to (\nu ab)(P\{^{a}/_{x}\} \mid Q\{^{b}/_{y}\})}$$

$$[\text{R-COM-SYNC-EXT}]\quad \frac{k \in I \quad \mathsf{e} \downarrow \mathsf{v}}{(\nu ab)(a!l_k\langle \mathsf{e}\rangle.P \mid \sum_{i\in I} b?l_i(x_i).Q_i) \to (\nu ab)(P \mid Q_k\{^{\mathsf{v}}/_{x_k}\})}$$

$$[\text{R-T-COND}]\quad \frac{\mathsf{e} \downarrow \texttt{true}}{\texttt{if } \mathsf{e} \texttt{ then } P \texttt{ else } Q \to P} \qquad [\text{R-F-COND}]\quad \frac{\mathsf{e} \downarrow \texttt{false}}{\texttt{if } \mathsf{e} \texttt{ then } P \texttt{ else } Q \to Q}$$

$$[\text{R-DEF-EXT}]\quad \frac{\tilde{\mathsf{e}} \downarrow \tilde{\mathsf{v}}}{\texttt{def } X(\tilde{x}\tilde{y}) = P \texttt{ in } (X\langle \tilde{\mathsf{e}}\tilde{a}\rangle \mid Q) \to \texttt{def } X(\tilde{x}\tilde{y}) = P \texttt{ in } (P\{^{\tilde{\mathsf{v}}}/_{\tilde{x}}\}\{^{\tilde{a}}/_{\tilde{y}}\} \mid Q)}$$

Table 19: Reduction of extended synchronous processes.

$$[\text{ERR-DEF}]\quad \frac{\tilde{\mathsf{e}} \not\downarrow}{X\langle \tilde{\mathsf{e}}\tilde{a}\rangle \to \texttt{error}} \qquad [\text{ERR-COM-EXT}]\quad \frac{\mathsf{e} \not\downarrow}{a!l\langle \mathsf{e}\rangle.P \to \texttt{error}} \qquad [\text{ERR-COND}]\quad \frac{\mathsf{e} \not\downarrow}{\texttt{if } \mathsf{e} \texttt{ then } P \texttt{ else } Q \to \texttt{error}}$$

$$[\text{ERR-CHAN-IN}]\quad \sum_{i\in I} \mathsf{v}?l_i(x_i).P_i \to \texttt{error} \qquad [\text{ERR-CHAN-OUT}]\quad \mathsf{v}!l\langle u'\rangle.P \to \texttt{error}$$

$$[\text{ERR-ACC}]\quad \frac{u = \mathsf{v} \text{ or } u = a}{u(x).P \to \texttt{error}} \qquad [\text{ERR-REQ}]\quad \frac{u = \mathsf{v} \text{ or } u = a}{\overline{u}(x).P \to \texttt{error}}$$

Table 20: Error reduction of extended synchronous processes.

Type system. *Sorts* (ranged over by B) and *extended session types* (ranged over by T) are defined by:

$$\begin{array}{lcl} B & ::= & \texttt{bool} \mid \texttt{nat} \mid \texttt{int} \mid <T> \\ T & ::= & \&_{i\in I} ?l_i(U_i).T_i \mid \bigoplus_{i\in I} !l_i\langle U_i\rangle.T_i \mid \mathbf{t} \mid \mu\mathbf{t}.T \mid \texttt{end} \\ U & ::= & B \mid T \end{array}$$

where $<T>$ is the sort of shared channels binding linear channels of extended session type T.

Subsorting $\leq:$ on sorts is the minimal reflexive and transitive closure of the relation induced by the rule: $\texttt{nat} \leq: \texttt{int}$. Table 21 gives the (expected) typing rules for expressions.

The synchronous subtyping rules for extended session types are obtained from the rules of Table 6 by replacing S with U, and by defining:

$$U \leqslant U' = \begin{cases} T \leqslant T' & \text{if } U = T \text{ and } U' = T', \\ B' \leq: B & \text{if } U = B \text{ and } U' = B', \\ \texttt{true} & \text{if } U = B \text{ and } U' = \texttt{end}, \\ \texttt{false} & \text{otherwise.} \end{cases}$$

$$\Gamma \vdash \mathtt{true} : \mathtt{bool} \qquad \Gamma \vdash \mathtt{false} : \mathtt{bool} \qquad \Gamma \vdash \mathsf{n} : \mathtt{nat} \qquad \Gamma \vdash \mathsf{i} : \mathtt{int} \qquad \Gamma, u : B \vdash u : B$$

$$\frac{\Gamma \vdash \mathsf{e} : \mathtt{bool}}{\Gamma \vdash \neg \mathsf{e} : \mathtt{bool}} \qquad \frac{\Gamma \vdash \mathsf{e} : \mathtt{nat}}{\Gamma \vdash \mathtt{succ}(\mathsf{e}) : \mathtt{nat}} \qquad \frac{\Gamma \vdash \mathsf{e} : \mathtt{int}}{\Gamma \vdash \mathtt{neg}(\mathsf{e}) : \mathtt{int}}$$

$$\frac{\Gamma \vdash \mathsf{e}_1 : \mathtt{int} \quad \Gamma \vdash \mathsf{e}_2 : \mathtt{int}}{\Gamma \vdash \mathsf{e}_1 > \mathsf{e}_2 : \mathtt{bool}} \qquad \frac{\Gamma \vdash \mathsf{e} : B \quad B \leq: B'}{\Gamma \vdash \mathsf{e} : B'}$$

Table 21: Typing rules for expressions.

$$[\text{T-ACC}]\ \frac{\Gamma, u :< T >\vdash P \triangleright \Delta, x : T}{\Gamma, u :< T >\vdash u(x).P \triangleright \Delta} \qquad [\text{T-REQ}]\ \frac{\Gamma, u :< T >\vdash P \triangleright \Delta, x : \overline{T}}{\Gamma, u :< T >\vdash \overline{u}(x).P \triangleright \Delta} \qquad [\text{T-RES}]\ \frac{\Gamma, \mathsf{s} :< T >\vdash P \triangleright \Delta}{\Gamma \vdash (\nu \mathsf{s})P \triangleright \Delta}$$

$$[\text{T-OUT-EXT}]\ \frac{\Gamma \vdash \mathsf{e} : B \quad \Gamma \vdash P \triangleright \Delta, u : T}{\Gamma \vdash u!l\langle \mathsf{e} \rangle.P \triangleright \Delta, u : !l\langle B \rangle.T} \qquad [\text{T-COND}]\ \frac{\Gamma \vdash \mathsf{e} : \mathtt{bool} \quad \Gamma \vdash P_1 \triangleright \Delta \quad \Gamma \vdash P_2 \triangleright \Delta}{\Gamma \vdash \mathtt{if}\ \mathsf{e}\ \mathtt{then}\ P_1\ \mathtt{else}\ P_2 \triangleright \Delta}$$

$$[\text{T-VAR-EXT}]\ \frac{\Gamma \vdash \tilde{\mathsf{e}} : \tilde{B}}{\Gamma, X : \langle \tilde{B}\tilde{T} \rangle \vdash X\langle \tilde{\mathsf{e}}\tilde{u} \rangle \triangleright \{\tilde{u} : \tilde{T}\}} \qquad [\text{T-DEF-EXT}]\ \frac{\Gamma, \tilde{x} : \tilde{B}, X : \langle \tilde{B}\tilde{T} \rangle \vdash P \triangleright \{\tilde{y} : \tilde{T}\} \qquad \Gamma, X : \langle \tilde{B}\tilde{T} \rangle \vdash Q \triangleright \Delta}{\Gamma \vdash \mathtt{def}\ X(\tilde{x}\tilde{y}) = P\ \mathtt{in}\ Q \triangleright \Delta}$$

Table 22: Typing rules for extended synchronous processes.

Notice that processes do not contain occurrences of linear channels typed by `end`, so any value can be sent to a process waiting for a linear channel typed by `end`. This justifies the value `true` in the definition of $U \leqslant U'$. Notice also that the extended session types and the sorts behave in opposite ways for inputs and outputs.

Shared environments associate identifiers to sorts and process variables to sequences of sorts and extended session types

$$\Gamma ::= \emptyset \mid \Gamma, u : B \mid \Gamma, X : \langle \tilde{B}\tilde{T} \rangle$$

The typing rules for extended synchronous processes are given in Tables 7 and 22.

Preciseness. The characteristic processes are the processes of Definitions 3.1 and 6.1. For `bool`, `nat` and `int` we use conditionals and the constructors $\neg$, `succ` and `neg` in order to test values. For the sorts of shared channels both input and output characteristic processes contain accept and request constructors. If S is different from T, then either S is not a subtype of T or vice versa. Therefore at least one of the session initialised by a shared channel of type $< S >$ and a shared channel of type $< T >$ will reduce to `error`, see the last case of the proof of Theorem 6.4.

Definition 6.1 (Characteristic extended synchronous processes)**.**

$$
\begin{array}{lcl}
\boldsymbol{P}_?(u,l,\texttt{bool},T) & \stackrel{\text{def}}{=} & u?l(x).\texttt{if } \neg x \texttt{ then } \boldsymbol{P}(u,T) \texttt{ else } \boldsymbol{P}(u,T) \\
\boldsymbol{P}_?(u,l,\texttt{nat},T) & \stackrel{\text{def}}{=} & u?l(x).\texttt{if } \texttt{succ}(x) > 0 \texttt{ then } \boldsymbol{P}(u,T) \texttt{ else } \boldsymbol{P}(u,T) \\
\boldsymbol{P}_?(u,l,\texttt{int},T) & \stackrel{\text{def}}{=} & u?l(x).\texttt{if } \texttt{neg}(x) > 0 \texttt{ then } \boldsymbol{P}(u,T) \texttt{ else } \boldsymbol{P}(u,T) \\
\boldsymbol{P}_?(u,l,<S>,T) & \stackrel{\text{def}}{=} & u?l(x).(\boldsymbol{P}(u,T) \mid x(y).\boldsymbol{P}(y,S) \mid \overline{x}(z).\boldsymbol{P}(z,\overline{S})) \\
\boldsymbol{P}_!(u,l,\texttt{bool},T) & \stackrel{\text{def}}{=} & u!l\langle\texttt{true}\rangle.\boldsymbol{P}(u,T) \\
\boldsymbol{P}_!(u,l,\texttt{nat},T) & \stackrel{\text{def}}{=} & u!l\langle 5\rangle.\boldsymbol{P}(u,T) \\
\boldsymbol{P}_!(u,l,\texttt{int},T) & \stackrel{\text{def}}{=} & u!l\langle -5\rangle.\boldsymbol{P}(u,T) \\
\boldsymbol{P}_!(u,l,<S>,T) & \stackrel{\text{def}}{=} & (\nu \mathsf{s})(u!l\langle \mathsf{s}\rangle.(\boldsymbol{P}(u,T) \mid \mathsf{s}(y).\boldsymbol{P}(y,S) \mid \overline{\mathsf{s}}(z).\boldsymbol{P}(z,\overline{S})))
\end{array}
$$

As for subtyping, the negation of extended synchronous subtyping is obtained from the rules of Table 8 by replacing S with U, and by defining:

$$
U \not\lhd U' = \begin{cases} T \not\lhd T' & \text{if } U = T \text{ and } U' = T', \\ B' \not\leq: B & \text{if } U = B \text{ and } U' = B', \\ \texttt{false} & \text{if } U = B \text{ and } U' = \texttt{end}, \\ \texttt{true} & \text{otherwise.} \end{cases}
$$

Lemmas 3.2 and 3.3 easily extend to these definitions, i.e. we get $\vdash_{\mathsf{s}} \boldsymbol{P}(u,T) \triangleright \{u : T\}$ and if $S \leqslant_{\mathsf{s}} T$ is not derivable, then $S \not\lhd_{\mathsf{s}} T$ is derivable. We are now ready to show preciseness.

Theorem 6.2 (Preciseness for extended synchronous subtyping)**.** *The extended synchronous subtyping relation is precise for the extended synchronous calculus.*

Proof. As in previous cases soundness follows from subject reduction, which can be easily proved. For completeness the only new cases are applications of rules [N-EXCH-BRA] and [N-EXCH-SEL] with sorts. We consider two paradigmatic cases here and other two paradigmatic cases in the proof of preciseness for the extended asynchronous calculus (Theorem 6.4).

Case [N-EXCH-BRA]: $T = \&_{i\in I}?l_i(U_i).T_i$, $S = \&_{j\in J}?l'_j(U'_j).S_j$, and $\exists k \in I\ \exists k' \in J$ such that $l_k = l'_{k'}$ and $U_k \not\lhd_{\mathsf{s}} U'_{k'}$. We only consider the case $U_k = \texttt{nat}$ and $U'_{k'} = \texttt{int}$.

$$
\begin{array}{l}
(\nu ab)(\boldsymbol{P}(a,T) \mid \boldsymbol{P}(b,\overline{S})) \rightarrow_{\mathsf{s}} \\
(\nu ab)(\sum_{i\in I\setminus\{k\}} \boldsymbol{P}_?(a,l_i,T'_i,T_i) \ +\ a?l_k(x).\texttt{if } \texttt{succ}(x) > 0 \texttt{ then } \boldsymbol{P}(a,T_k) \texttt{ else } \boldsymbol{P}(a,T_k) \mid \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad b!l_k\langle -5\rangle.\boldsymbol{P}(b,\overline{S_{k'}})) \rightarrow_{\mathsf{s}} \\
C[\texttt{if } \texttt{succ}(-5) > 0 \texttt{ then } \boldsymbol{P}(a,T_k) \texttt{ else } \boldsymbol{P}(a,T_k)]
\end{array}
$$

where $C[\ ] = (\nu ab)(\boldsymbol{P}(b,\overline{S_{k'}}) \mid [\ ])$. Being $\texttt{succ}(-5) > 0$ stuck, by rule [ERR-COND]

$$\texttt{if } \texttt{succ}(-5) > 0 \texttt{ then } \boldsymbol{P}(a,T_k) \texttt{ else } \boldsymbol{P}(a,T_k) \rightarrow_{\mathsf{s}} \texttt{error}$$

then by rule [ERR-CONTEXT], we conclude

$$C[\texttt{if } \texttt{succ}(-5) > 0 \texttt{ then } \boldsymbol{P}(a,T_k) \texttt{ else } \boldsymbol{P}(a,T_k)] \rightarrow_{\mathsf{s}} \texttt{error}$$

$$\text{[R-INIT-ASYNC]}\quad \frac{a, b \text{ fresh}}{\mathsf{s}(x).P \mid \overline{\mathsf{s}}(y).Q \to (\nu ab)(P\{^{a}/_{x}\} \mid Q\{^{b}/_{y}\} \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing)}$$

$$\text{[R-SEND-ASYNC-EXT]}\quad \frac{\mathsf{e} \downarrow \mathsf{v}}{ab \blacktriangleright h \mid a!l\langle \mathsf{e} \rangle.P \to ab \blacktriangleright h \cdot l\langle \mathsf{v} \rangle \mid P} \qquad \text{[R-RECEIVE-ASYNC-EXT]}\quad \frac{k \in I}{ab \blacktriangleright l_k\langle \mathsf{v} \rangle \cdot h \mid \sum_{i \in I} b?l_i(x_i).P_i \to ab \blacktriangleright h \mid P_k\{^{\mathsf{v}}/_{x_k}\}}$$

Table 23: Reduction of asynchronous processes.

Case [N-EXCH-SEL]: $T = \bigoplus_{i \in I} !l_i\langle U_i \rangle.T_i$, $S = \bigoplus_{j \in J} !l'_j\langle U'_j \rangle.S_j$, and $\exists k \in I\ \exists k' \in J$ such that $l_k = l'_{k'}$ and $U'_{k'} \not\leqslant_{\mathsf{s}} U_k$. We only consider the case $U_k = \texttt{bool}$ and $U'_{k'} = T' \neq \texttt{end}$.

$(\nu ab)(\boldsymbol{P}(a, T) \mid \boldsymbol{P}(b, \overline{S})) \to_{\mathsf{s}}$
$(\nu ab)(a!l_k\langle \texttt{true} \rangle.\boldsymbol{P}(a, T_k) \mid \sum_{j \in J \setminus \{k'\}} \boldsymbol{P}_?(b, l'_j, U'_j, \overline{S_j}) + b?l_k(x).(\boldsymbol{P}(b, \overline{S_{k'}}) \mid \boldsymbol{P}(x, T'))) \to_{\mathsf{s}}$
$C[\boldsymbol{P}(\texttt{true}, T')]$

where $C[\ ] = (\nu ab)(\boldsymbol{P}(a, T_k) \mid \boldsymbol{P}(b, \overline{S_{k'}}) \mid [\ ])$. By definition of characteristic process, being $T' \neq \texttt{end}$, the value $\texttt{true}$ is used as a channel in $\boldsymbol{P}(\texttt{true}, T')$, and this implies by rule [ERR-CHAN-IN] or [ERR-CHAN-OUT]

$$\boldsymbol{P}(\texttt{true}, T') \to_{\mathsf{s}} \texttt{error}$$

then by rule [ERR-CONTEXT], we conclude

$$C[\boldsymbol{P}(\texttt{true}, T')] \to_{\mathsf{s}} \texttt{error}$$

□

6.2. **Asynchronous Communication.**

Syntax and operational semantics. The processes of the extended asynchronous session calculus are generated by the rules of Table 9, where messages can be of the form $l\langle \mathsf{v} \rangle$, and by the rules of Table 17.

The rules for evaluating expressions remain those of Table 18. Structural congruence and evaluation contexts are generalised in the obvious way. The reduction rules for the extended asynchronous processes are obtained from the reduction rules of asynchronous processes of § 4 by adding the rules [R-T-COND], [R-F-COND], [R-DEF-EXT] of Table 19 and the rules of Table 23. The mapping δ is extended in the obvious way wrt. Table 12 — i.e., adding the cases:

$$\delta(X\langle \tilde{\mathsf{e}}\tilde{u} \rangle, \tilde{D}, \chi) = \begin{cases} \delta(P\{^{\tilde{\mathsf{e}}\tilde{u}}/_{\tilde{x}\tilde{y}}\}, \tilde{D}, \chi \cdot X\langle \tilde{\mathsf{e}}\tilde{u} \rangle) & \text{if } X\langle \widetilde{\mathsf{e}_0}\tilde{u} \rangle \notin \chi \text{ and} \\ & X\langle \tilde{x}\tilde{y} \rangle = P \in \tilde{D} \\ \emptyset & \text{otherwise} \end{cases}$$

$$\delta(u(x).P, \tilde{D}, \chi) = \delta(\overline{u}(x).P, \tilde{D}, \chi) = \delta(P, \tilde{D}, \chi) \setminus \{x\}$$
$$\delta(u!l\langle \mathsf{e} \rangle.P, \tilde{D}, \chi) = \delta((\nu \mathsf{s})P, \tilde{D}, \chi) = \delta(P, \tilde{D}, \chi)$$
$$\delta(\texttt{if } \mathsf{e} \texttt{ then } P_1 \texttt{ else } P_2, \tilde{D}, \chi) = \delta(P_1, \tilde{D}, \chi) \cup \delta(P_2, \tilde{D}, \chi)$$

Notice that $\delta(X\langle \tilde{\mathsf{e}}\tilde{u} \rangle, \tilde{D}, \chi) = \emptyset$ if χ contains $X\langle \widetilde{\mathsf{e}_0}\tilde{u} \rangle$ for some $\widetilde{\mathsf{e}_0}$, which can be different from $\tilde{\mathsf{e}}$.

The error reduction rules for extended asynchronous processes are obtained from the error reduction rules of asynchronous processes of § 4 by adding the rules of Table 20 and the rule

$$[\text{ERR-MISM-ASYNC-EXT}]\quad \frac{\forall i \in I : l \neq l_i}{ab \blacktriangleright l\langle u\rangle \cdot h \mid \sum_{i\in I} b?l_i(x_i).P_i \to \texttt{error}}$$

Type system. We define sorts, extended session types and shared environments as in § 6.1. The typing rules for expressions remain those of Table 21. The queue types are those of § 4.4 by replacing S with U, thus allowing $l\langle B\rangle$. Also the definition of session remainder is obtained from that of § 4.4 by replacing S with U.

The asynchronous subtyping for extended session types is obtained by defining $U \leqslant U'$ as in § 6.1 and replacing S with U in the subtyping of § 4.3. The typing rules for extended asynchronous processes are obtained from the typing rules of asynchronous processes of § 4.4 by adding the rules of Table 22 and the following rule:

$$[\text{T-MESSAGE-Q-V}]\quad \frac{\Gamma \vdash \mathsf{v} : B \quad \Gamma \vdash ba \blacktriangleright h \triangleright \Delta, ba : \tau}{\Gamma \vdash ba \blacktriangleright h \cdot l\langle \mathsf{v}\rangle \triangleright \Delta, ba : \tau \cdot l\langle B\rangle}$$

The reduction of extended asynchronous session environments is given by the rules of Table 15 after the replacement of S with U.

Preciseness. The characteristic processes are the processes of Definition 5.1 plus the processes of Definition 6.1. The negation of asynchronous subtyping is obtained from the negation of Section 5 by replacing S with U and by defining $U \not\leqslant U'$ as in 6.1.

Lemmas 5.2 and 5.3 easily extend to these definitions, i.e. we get $\vdash_{\mathsf{a}} \boldsymbol{P}(u, T) \triangleright \{u : T\}$ and if $S \leqslant_{\mathsf{a}} T$ is not derivable, then $S \not\leqslant_{\mathsf{a}} T$ is derivable. The invariance of the types of shared channels is intriguing in the proof of completeness. We need an auxiliary lemma.

Lemma 6.3. (1) *If $T \not\leqslant_{\mathsf{a}} S$ and $S \not\leqslant_{\mathsf{a}} T$, then one of the two statements is derivable without using rules* [N-BRA-ASYNC] *and* [N-SEL-ASYNC].
(2) *If $T \leqslant_{\mathsf{a}} S$ and $T \neq S$, then $S \not\leqslant_{\mathsf{a}} T$ is derivable without using rules* [N-BRA-ASYNC] *and* [N-SEL-ASYNC].

Proof. (1). Let assume we use [N-BRA-ASYNC] in the proof of $T \not\leqslant_{\mathsf{a}} S$. Then there is a subderivation of $T \not\leqslant_{\mathsf{a}} S$ such that [N-BRA-ASYNC] is used to prove $T' \not\leqslant_{\mathsf{a}} S'$ or $S' \not\leqslant_{\mathsf{a}} T'$. Let $T' \not\leqslant_{\mathsf{a}} S'$ be consequence of an application of rule [N-BRA-ASYNC]. We have two cases:

- an application of rule [N-CONT-ASYNC] with a non-empty asynchronous context occurs in the derivation branch from $T' \not\leqslant_{\mathsf{a}} S'$ to $T \not\leqslant_{\mathsf{a}} S$. Let $T'' \not\leqslant_{\mathsf{a}} S''$ be the conclusion of the application of rule [N-CONT-ASYNC] (with a non-empty asynchronous context) that is *closest* to $T \not\leqslant_{\mathsf{a}} S$. Notice that the trees of T'', S'' are subtrees of those of T, S, respectively. We can then prove $S \not\leqslant_{\mathsf{a}} T$ with a subderivation which shows $S'' \not\leqslant_{\mathsf{a}} T''$ using rule [N-BRASEL];
- otherwise, the derivation branch from $T' \not\leqslant_{\mathsf{a}} S'$ to $T \not\leqslant_{\mathsf{a}} S$ does *not* contain any application of [N-CONT-ASYNC] with a non-empty asynchronous context. In this case, the trees of T', S' are subtrees of those of T, S, respectively. Let $\& \notin T'$ and $S' = \&_{i\in I} ?l_i(S_i).T_i$. Then T' is either `end` or a selection. We can prove $S \not\leqslant_{\mathsf{a}} T$ with a subderivation which shows

$S' \not\lhd_{\mathsf{a}} T'$. Taking into account the shapes of T', S' the only applicable rules are either [N-END R] or [N-BRASEL].

The proof when we use [N-SEL-ASYNC] in the derivation of $T \not\lhd_{\mathsf{a}} S$ is similar.

(2). The assumption $T \leqslant_{\mathsf{a}} S$ assures that T,S have corresponding branchings and selections. Hence, rules [N-BRA-ASYNC] and [N-SEL-ASYNC] cannot be used in showing $S \not\lhd_{\mathsf{a}} T$. □

Theorem 6.4 (Preciseness for extended asynchronous subtyping)**.** *The extended asynchronous subtyping relation is precise for the extended asynchronous calculus.*

Proof. As in previous cases soundness follows from subject reduction, which can be easily proved. For completeness let $T \not\lhd_{\mathsf{a}} S$. We show by induction on $\not\lhd_{\mathsf{a}}$:

(1) if $T \not\lhd_{\mathsf{a}} S$ is derivable without using rules [N-BRA-ASYNC] and [N-SEL-ASYNC], then

$$(\nu ab)(\boldsymbol{P}(a,T) \mid \boldsymbol{P}(b,\overline{S}) \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing) \rightarrow_{\mathsf{a}}^{*} \mathtt{error}$$

(2) otherwise there are $T' \leqslant_{\mathsf{s}} T$, $S' \leqslant_{\mathsf{s}} \overline{S}$ with either $T' = T$ or $S' = \overline{S}$, such that

$$(\nu ab)(\boldsymbol{P}(a,T') \mid \boldsymbol{P}(b,S') \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing) \rightarrow_{\mathsf{a}}^{*} \mathtt{error}$$

It is easy to check that the proof of Theorem 5.7 shows this stronger statement, since we need to consider subtypes of the current types only dealing with rules [N-BRA-ASYNC] and [N-SEL-ASYNC].

We consider here only two cases of applications of rules [N-EXCH-BRA] and [N-EXCH-SEL].

Case [N-EXCH-BRA]: $T = \&_{i\in I}?l_i(U_i).T_i$, $S = \&_{j\in J}?l'_j(U'_j).S_j$, and $\exists k \in I\ \exists k' \in J : l_k = l'_{k'}$ such that $U_k \not\lhd_{\mathsf{a}} U'_{k'}$. We only consider the case $U'_k = T' \neq \mathtt{end}$ and $U'_{k'} =< S' >$.

$(\nu ab)(\boldsymbol{P}(a,T) \mid \boldsymbol{P}(b,\overline{S}) \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing) \rightarrow_{\mathsf{a}}$
$(\nu ab)(\boldsymbol{P}(a,T) \mid (\nu \mathsf{s})(b!l_k\langle \mathsf{s}\rangle.(\boldsymbol{P}(b,\overline{S_{k'}}) \mid \mathsf{s}(y).\boldsymbol{P}(y,S') \mid \overline{\mathsf{s}}(z).\boldsymbol{P}(z,\overline{S'}))) \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing) \rightarrow_{\mathsf{a}}$
$(\nu ab)(\nu \mathsf{s})(\sum_{i\in I} \boldsymbol{P}_?(a,l_i,T'_i,T_i) \mid \boldsymbol{P}(b,\overline{S_{k'}}) \mid \mathsf{s}(y).\boldsymbol{P}(y,S') \mid \overline{\mathsf{s}}(z).\boldsymbol{P}(z,\overline{S'}) \mid ba \blacktriangleright l_k\langle \mathsf{s}\rangle \mid ab \blacktriangleright \varnothing) \rightarrow_{\mathsf{a}}$
$C[\boldsymbol{P}(\mathsf{s},T')]$

where $C[\ ] = (\nu ab)(\nu \mathsf{s})(\boldsymbol{P}(a,T_k) \mid \boldsymbol{P}(b,\overline{S_{k'}}) \mid \mathsf{s}(y).\boldsymbol{P}(y,S') \mid \overline{\mathsf{s}}(z).\boldsymbol{P}(z,\overline{S'}) \mid [\ ] \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing)$. By definition of characteristic process, being $T' \neq \mathtt{end}$, the shared channel s is used as a linear channel in $\boldsymbol{P}(\mathsf{s},T')$, and this implies by rule [ERR-CHAN-IN] or [ERR-CHAN-OUT]

$$\boldsymbol{P}(\mathsf{s},T') \rightarrow_{\mathsf{a}} \mathtt{error}$$

then by rule [ERR-CONTEXT], we conclude

$$C[\boldsymbol{P}(\mathsf{s},T')] \rightarrow_{\mathsf{a}} \mathtt{error}$$

Case [N-EXCH-SEL]: $T = \bigoplus_{i\in I} !l_i\langle U_i\rangle.T_i$, $S = \bigoplus_{j\in J} !l'_j\langle U'_j\rangle.S_j$, and $\exists k \in I\ \exists k' \in J$ such that $l_k = l'_{k'}$ and $U'_{k'} \not\lhd_{\mathsf{a}} U_k$. We only consider the case $U_k =< T' >$ and $U'_{k'} =< S' >$.

$(\nu ab)(\boldsymbol{P}(a,T) \mid \boldsymbol{P}(b,\overline{S}) \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing) \rightarrow_{\mathsf{a}}$
$(\nu ab)((\nu \mathsf{s})(a!l_k\langle \mathsf{s}\rangle.(\boldsymbol{P}(a,T_k) \mid \mathsf{s}(y).\boldsymbol{P}(y,T') \mid \overline{\mathsf{s}}(z).\boldsymbol{P}(z,\overline{T'}))) \mid \boldsymbol{P}(b,\overline{S}) \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing) \rightarrow_{\mathsf{a}}$
$(\nu ab)(\nu \mathsf{s})(\boldsymbol{P}(a,T_k) \mid \mathsf{s}(y).\boldsymbol{P}(y,T') \mid \overline{\mathsf{s}}(z).\boldsymbol{P}(z,\overline{T'}) \mid \sum_{j\in J\setminus\{k'\}} \boldsymbol{P}_?(b,l'_j,U'_i,S_j)$
$\quad + b?l_k(x).(\boldsymbol{P}(b,\overline{S_{k'}}) \mid x(y').\boldsymbol{P}(y',S') \mid \overline{x}(z').\boldsymbol{P}(z',\overline{S'})) \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright l_k\langle \mathsf{s}\rangle) \rightarrow_{\mathsf{a}}$
$(\nu ab)(\nu \mathsf{s})(\boldsymbol{P}(a,T_k) \mid \mathsf{s}(y).\boldsymbol{P}(y,T') \mid \overline{\mathsf{s}}(z).\boldsymbol{P}(z,\overline{T'}) \mid$
$\quad \boldsymbol{P}(b,\overline{S_{k'}}) \mid \mathsf{s}(y').\boldsymbol{P}(y',S') \mid \overline{\mathsf{s}}(z').\boldsymbol{P}(z',\overline{S'})) \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing)$

Since T' and S' cannot be equal, then we can derive either $S' \not\triangleleft_{\mathsf{a}} T'$ or $T' \not\triangleleft_{\mathsf{a}} S'$ with a proof which does not use rules [N-BRA-ASYNC] and [N-SEL-ASYNC] by Lemma 6.3. In the first case the obtained process can be written as $C[\overline{\mathsf{s}}(z).\boldsymbol{P}(z, \overline{T'}) \mid \mathsf{s}(y').\boldsymbol{P}(y', S')]$ and

$$C[\overline{\mathsf{s}}(z).\boldsymbol{P}(z, \overline{T'}) \mid \mathsf{s}(y').\boldsymbol{P}(y', S')] \to_{\mathsf{a}} C[(\nu cd)(\boldsymbol{P}(c, \overline{T'}) \mid \boldsymbol{P}(d, S') \mid dc \blacktriangleright \varnothing \mid cd \blacktriangleright \varnothing)]$$

By above we are in case (1) of induction, therefore we get

$$(\nu cd)(\boldsymbol{P}(c, \overline{T'}) \mid \boldsymbol{P}(d, S') \mid dc \blacktriangleright \varnothing \mid cd \blacktriangleright \varnothing) \to_{\mathsf{a}}^{*} \mathtt{error}$$

then by rule [ERR-CONTEXT]

$$C[(\nu cd)(\boldsymbol{P}(c, \overline{T'}) \mid \boldsymbol{P}(d, S') \mid dc \blacktriangleright \varnothing \mid cd \blacktriangleright \varnothing)] \to_{\mathsf{a}}^{*} \mathtt{error}$$

In the second case the obtained process can be written as $C'[\mathsf{s}(y).\boldsymbol{P}(y, T') \mid \overline{\mathsf{s}}(z').\boldsymbol{P}(z', \overline{S'})]$ and we conclude similarly. □

7. Denotational Preciseness

In λ-calculus types are usually interpreted as subsets of the domains of λ-models [3, 32]. *Denotational preciseness* of subtyping is then:

$$T \leqslant S \quad \text{if and only if} \quad [\![T]\!] \subseteq [\![S]\!]$$

using $[\![\ ]\!]$ to denote type interpretation.

In the present context let us interpret a session type T as the set of processes with only one free channel typed by T, i.e.

$$[\![T]\!]_{*} = \{P \ \mid \vdash_{*} P \triangleright \{a : T\}\}$$

where $* \in \{\mathsf{s}, \mathsf{a}\}$. We can then show that both the synchronous and the asynchronous subtypings are denotationally precise. Rule [T-SUB] gives the denotational soundness. Denotational completeness follows from the following key property of characteristic processes:

$$\vdash_{*} \boldsymbol{P}_{*}(a, T) \triangleright \{a : S\} \quad \text{implies} \quad T \leqslant_{*} S \tag{7.1}$$

where $\boldsymbol{P}_{*}(a, T)$ is the synchronous characteristic process if $* = \mathsf{s}$ and the asynchronous characteristic process if $* = \mathsf{a}$. The property (7.1) can be shown by induction on T using Inversion Lemmas for synchronous and asynchronous processes (Lemma A.2 in Appendix A and Lemma B.4 in Appendix B).

If $T \not\leqslant_{*} S$, then $\boldsymbol{P}_{*}(a, T) \in [\![T]\!]_{*}$, but $\boldsymbol{P}_{*}(a, T) \notin [\![S]\!]_{*}$, which implies $[\![T]\!]_{*} \not\subseteq [\![S]\!]_{*}$. Therefore we get denotational completeness.

Theorem 7.1 (Denotational preciseness)**.** *The synchronous and the asynchronous subtyping relations are denotationally precise for the synchronous calculus and the asynchronous calculus, respectively.*

To sum up, the existence of characteristic processes implies denotational preciseness when each session type T is interpreted as the set of processes having only one channel of type T.

8. Related Work

Preciseness. To the best of our knowledge, operational preciseness was first defined by Ligatti *et al.* [44], for a call-by-value λ-calculus with recursive functions, pairs and sums. In that paper the authors show that the iso-recursive subtyping induced by the Amber rules [8] is incomplete. They propose a new iso-recursive subtyping which they prove to be operationally precise. Denotational preciseness of this subtyping has been recently proved [20].

Both operational and denotational preciseness are shown by Dezani-Ciancaglini and Ghilezan [18] for the concurrent λ-calculus with intersection and union types introduced by Dezani-Ciancaglini *et al.* [17]. In that paper divergence plays the rôle of reduction to `error`.

Preciseness in concurrency is more useful and challenging than in the functional setting, since there are many interesting choices for the syntax, semantics, type errors of the calculi and for the typing systems. A similar situation appears in the study of bisimulations, where many labelled transition relations can be defined. It is now common that researchers justify the correctness of labelled transition systems by proving that the bisimulation coincides with the contextual congruence [37, 47]. Our claim is that preciseness should become a sanity check for subtypings.

Recently preciseness has been shown for a synchronous multiparty session calculus without delegation [19].

Choices of typing system and subtyping. The first branching-selection subtyping for session types was proposed by Gay and Hole [23, 24] and has been used in other works by various authors [9, 11, 10, 26, 53, 54, 62]. Their approach corresponds to safe substitutability of *channels* (rather than *processes*): as a consequence, their subtyping is the opposite of $\leqslant_{\mathsf{s}}$, since branching is covariant and selection is controvariant in the set of labels; coherently, such a co/contra-variance is also embodied in their typing system and judgements.
A subtyping relation with the opposite direction has been used by Honda, Yoshida, Mostrous and other authors [7, 15, 48, 49, 50, 51]: their approach corresponds to safe substitutability of *processes*. An insightful comparison between these subtypings is the argument of a recent paper [22].

In this work, we have adopted the subtyping direction and typing system of Honda *et al.*, since they directly match a definition of preciseness based on process substitution (Definition 1.1), and thus allow for direct reasoning on characteristic processes. Establishing the preciseness of the Gay and Hole subtyping with respect to their typing system is less immediate, since the notion of characteristic process (and their substitution) needs to be adapted to their setting. However, since their typing rules "mirror" ours, and their subtyping "mirrors" $\leqslant_{\mathsf{s}}$, similar preciseness results can be proved by reversing the ordering both in the preciseness definition and in the extension of subtyping to session environments.

Other completeness results. Subtyping of recursive types requires algorithms for checking subtype relations, as discussed by Pierce [56, § 21]. These algorithms need to be proved sound and complete with respect to the definition of the corresponding subtyping, as done in several works [10, 24, 57]. Synchronous subtyping can be easily decided, see for example [24]. We leave the development of an algorithm for asynchronous subtyping, and the proof of its soundness and completeness, as future work.

Several works on subtyping formulate the errors using typed reductions or type environments [30, 57], and they prove soundness with respect to the typed reductions and their erasure theorems. In contrast with these approaches, our error definitions in Tables 4 and 13 do not rely on any type-case construct or explicit type information, but are defined *syntactically* over untyped terms. Note that once the calculus is annotated by type information or equipped with type case, completeness becomes trivial, since any two processes of incomparable types can be operationally distinguished.

Semantic subtyping. In the semantic subtyping approach by Frisch *et al.* [21], each type is interpreted as the set of values having that type and subtyping is subset inclusion between type interpretations. This gives a precise subtyping as soon as the calculus allows to distinguish operationally values of different types. Semantic subtyping has been studied by Castagna, De Nicola *et al.* [9] for a π-calculus with a patterned input, and by Castagna, Dezani-Ciancaglini *et al.* [10] for a session calculus with internal and external choices and typed input. Types are built using a rich set of type constructors including union, intersection and negation: they extend IO-types in Castagna, De Nicola *et al.* [9], and session types in Castagna, Dezani-Ciancaglini *et al.* [10]. Semantic subtyping is precise for the calculi of all such works, thanks to the type case constructor in the work by Frisch *et al.* [21], and to the blocking of inputs for values of "wrong" types in the works by Castagna *et al.* [9, 10].

Subtyping of Mostrous PhD thesis. Our subtyping relation differs from that defined in Mostrous thesis [48, 50] for the premises $\& \in \mathcal{A}$ and $\& \in T_i$ in rule [SUB-PERM-ASYNC]. As a consequence in that thesis T is a subtype of S when $T = \mu\mathbf{t}.!l\langle T'\rangle.\mathbf{t}$ and $S = \mu\mathbf{t}.!l\langle T'\rangle.?l'(S').\mathbf{t}$ (see p. 116 of Mostrous thesis). This subtyping is not sound in our system: intuitively T accumulates infinite orphan messages in a queue, while S ensures that the messages are eventually received. The subtyping relation in Mostrous thesis unexpectedly allows an unsound process (typed by T) to act as if it were a sound process (typed by S). Let $C = (\nu ab)([\,] \mid Q \mid ab \blacktriangleright \varnothing \mid ba \blacktriangleright \varnothing)$ where

$$Q = \mathtt{def}\ Y(x) = b!l\langle x\rangle.b?l'(y).Y\langle x\rangle\ \mathtt{in}\ (\nu cc')(Y\langle c\rangle)$$

Then we can derive $C[a : S] \triangleright \emptyset$. Let

$$P = \mathtt{def}\ Z(z) = a!l\langle z\rangle.Z\langle z\rangle\ \mathtt{in}\ (\nu dd')(Z\langle d\rangle)$$

Then $P \triangleright \{a : T\}$. We get

$$C[P] \to_{\mathsf{a}}^{*} (\nu ab)(\nu cc')(P \mid Q \mid ab \blacktriangleright \varnothing \mid ba \blacktriangleright l\langle c\rangle) \to_{\mathsf{a}} \mathtt{error}$$

by rule [ERR-ORPH-MESS-ASYNC], since $a \notin \varphi(P \mid Q \mid ab \blacktriangleright \varnothing)$ and $\mathsf{fpv}(P \mid Q \mid ab \blacktriangleright \varnothing) = \emptyset$.

The subtyping of Mostrous thesis is sound for the session calculus defined there, which does not consider orphan messages as errors. However, the subtyping of Mostrous thesis is not complete, an example being $\mu\mathbf{t}.!l\langle T\rangle.\mathbf{t} \not\leqslant \mu\mathbf{t}.?l'(S).\mathbf{t}$. There is no context C which is safe for all processes with one channel typed by $\mu\mathbf{t}.?l'(S).\mathbf{t}$ and no process P with one channel typed by $\mu\mathbf{t}.!l\langle T\rangle.\mathbf{t}$ such that $C[P]$ deadlocks.

9. Conclusion

This article gives, as far as we know, the first formulation and proof techniques for the preciseness of subtyping in mobile processes. We consider the synchronous and asynchronous session calculi to investigate the preciseness of the existing subtypings. While the well-known branching-selection subtyping [7, 15, 24] is precise for the synchronous calculus, the subtyping in Mostrous thesis [48, 50] turns out to be not sound for the asynchronous calculus. We propose a simplification of previous asynchronous subtypings [49, 51] and prove its preciseness. As a matter of fact only soundness is a consequence of subject reduction, while completeness can fail also when subject reduction holds.

The formulation of preciseness along with the proof methods and techniques could be useful to examine other subtypings and calculi. Our future work includes the applications to higher-order processes [48, 49, 50], polymorphic types [5, 26, 28], fair subtypings [53, 54] and contract subtyping [1]. We plan to use the characteristic processes in typecheckers for session types. More precisely the error messages can show processes of given types when type checking fails. One interesting problem is to find the necessary and sufficient conditions to obtain completeness of the generic subtyping by Igarashi and Kobayashi [41]. Such a characterisation would give preciseness for the many type systems which are instances of generic subtyping [41]. The notion of subtyping is clearly connected with that of type duality. Various definitions of dualities are compared by Bernardi *et al.* [4], and we plan to investigate if completeness of subtyping can be used in finding the largest safe duality.

The recent study on the Curry-Howard isomorphism between session types and Linear Logic [5, 6, 55, 65] gives a logic basis to session type duality. Also the one-to-one correspondence between session types and a class of deadlock-free communicating automata [29] has shown that session types have solid roots [16, 64].

The preciseness result for the synchronous calculus in § 2 and § 3 shows a rigorousness of the branching-selection subtyping, which is implemented (as a default) in most of session-based programming languages and tools [7, 15, 31, 35, 39, 40, 60] for enlarging typeability. For the asynchronous calculus, preciseness is more debatable since it depends on the choice of type safety properties, see § 4 and § 5. But in this case preciseness plays a more important rôle, since a programmer can adjust a subtyping relation to loosen or tighten subtypings with respect to the type safety properties which she wishes to guarantee. Once preciseness has been proved, she can be sure that her safety specifications and the subtyping have an exact match with respect to both static and dynamic semantics.

Acknowledgments. We are grateful to the anonymous reviewers for their useful suggestions, which led to substantial improvements. We are indebted to Jovanka Pantović for pointing out a subtle mistake in a previous version of the completeness proof.

References

[1] Franco Barbanera and Ugo de Liguoro. Two notions of sub-behaviour for session-based client/server systems. In *PPDP*, pages 155–164. ACM Press, 2010.

[2] Henk Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North Holland, 1984.

[3] Henk Barendregt, Mario Coppo, and Mariangiola Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *Journal of Symbolic Logic*, 48(4):931–940, 1983.

[4] Giovanni Bernardi, Ornela Dardha, Simon J. Gay, and Dimitrios Kouzapas. On duality relations for session types. In *TGC*, volume 8902 of *LNCS*, pages 51–66. Springer, 2014.

[5] Luís Caires, Jorge A. Pérez, Frank Pfenning, and Bernardo Toninho. Behavioral polymorphism and parametricity in session-based communication. In *ESOP*, volume 7792 of *LNCS*, pages 330–349. Springer, 2013.
[6] Luís Caires and Frank Pfenning. Session types as intuitionistic linear propositions. In *CONCUR*, volume 6269 of *LNCS*, pages 222–236. Springer, 2010.
[7] Marco Carbone, Kohei Honda, and Nobuko Yoshida. Structured communication-centered programming for web services. *ACM Transactions on Programming Languages and Systems*, 34(2):8:1–8:78, 2012.
[8] Luca Cardelli. Amber. In *Combinators and functional programming languages*, volume 242 of *LNCS*, pages 21–47. Springer, 1986.
[9] Giuseppe Castagna, Rocco De Nicola, and Daniele Varacca. Semantic subtyping for the pi-calculus. *Theoretical Computer Science*, 398(1-3):217–242, 2008.
[10] Giuseppe Castagna, Mariangiola Dezani-Ciancaglini, Elena Giachino, and Luca Padovani. Foundations of session types. In *PPDP*, pages 219–230. ACM Press, 2009.
[11] Giuseppe Castagna, Nils Gesbert, and Luca Padovani. A theory of contracts for web services. In *POPL*, pages 261–272. ACM Press, 2008.
[12] Gérard Cécé and Alain Finkel. Verification of programs with half-duplex communication. *Information and Computation*, 202(2):166–190, 2005.
[13] Tzu-Chun Chen, Mariangiola Dezani-Ciancaglini, and Nobuko Yoshida. On the preciseness of subtyping in session types. In *PPDP*, pages 135–146. ACM Press, 2014.
[14] Mario Coppo, Mariangiola Dezani-Ciancaglini, and Nobuko Yoshida. Asynchronous session types and progress for object-oriented languages. In *FMOODS*, volume 4468 of *LNCS*, pages 1–31. Springer, 2007.
[15] Romain Demangeon and Kohei Honda. Full abstraction in a subtyped pi-calculus with linear types. In *CONCUR*, volume 6901 of *LNCS*, pages 280–296. Springer, 2011.
[16] Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty compatibility in communicating automata: Characterisation and synthesis of global session types. In *ICALP*, volume 7966 of *LNCS*, pages 174–186. Springer, 2013.
[17] Mariangiola Dezani-Ciancaglini, Ugo de'Liguoro, and Adolfo Piperno. A filter model for concurrent lambda-calculus. *SIAM Journal on Computing*, 27(5):1376–1419, 1998.
[18] Mariangiola Dezani-Ciancaglini and Silvia Ghilezan. Preciseness of subtyping on intersection and union types. In *RTATLCA*, volume 8560 of *LNCS*, pages 194–207. Springer, 2014.
[19] Mariangiola Dezani-Ciancaglini, Silvia Ghilezan, Svetlana Jaksic, Jovanka Pantovic, and Nobuko Yoshida. Precise subtyping for synchronous multiparty sessions. In *PLACES*, volume 203 of *EPTCS*, pages 29 – 44, 2015.
[20] Mariangiola Dezani-Ciancaglini, Silvia Ghilezan, Jovanka Pantovic Svetlana Jaksic, and Nobuko Yoshida. Denotational and operational preciseness of subtyping: A roadmap. In *Theory and Practice of Formal Methods*, volume 9660 of *LNCS*, pages 155–172. Springer-Verlag, 2016.
[21] Alain Frisch, Giuseppe Castagna, and Veronique Benzaken. Semantic subtyping: dealing set-theoretically with function, union, intersection, and negation types. *Journal of ACM*, 55(4):1–64, 2008.
[22] Simon Gay. Subtyping supports safe session substitution. In *A List of Successes that can Change the World: Essays in Honour of Phil Wadler*, volume 9600 of *LNCS*, pages 95–108, 2016.
[23] Simon Gay and Malcolm Hole. Types and Subtypes for Client-Server Interactions. In *ESOP*, volume 1576 of *LNCS*, pages 74–90. Springer-Verlag, 1999.
[24] Simon Gay and Malcolm Hole. Subtyping for session types in the pi calculus. *Acta Informatica*, 42(2/3):191–225, 2005.
[25] Simon Gay and Vasco T. Vasconcelos. Linear type theory for asynchronous session types. *Journal of Functional Programming*, 20(1):19–50, 2010.
[26] Simon J. Gay. Bounded polymorphism in session types. *Mathematical Structures in Computer Science*, 18(5):895–930, 2008.
[27] Simon J. Gay, Vasco Thudichum Vasconcelos, António Ravara, Nils Gesbert, and Alexandre Z. Caldeira. Modular session types for distributed object-oriented programming. In *POPL*, pages 299–312. ACM Press, 2010.
[28] Matthew A. Goto, Radha Jagadeesan, Alan Jeffrey, Corin Pitcher, and James Riely. An extensible approach to session polymorphism. *Mathematical Structures in Computer Science*, 26(3):465–509, 2016.
[29] M. Gawdat Gouda, E. George Manning, and Yong Yu. On the progress of communication between two finite state machines. *Information and Control*, 63:200–216, 1984.

[30] Matthew Hennessy and James Riely. Resource access control in systems of mobile agents. *Information and Computation*, 173(1):82–120, 2002.
[31] Anders S. Henriksen, Lasse Nielsen, Thomas T. Hildebrandt, Nobuko Yoshida, and Fritz Henglein. Trustworthy pervasive healthcare services via multi-party session types. In *FHIES*, volume 7789 of *LNCS*, pages 124–141. Springer, 2012.
[32] J. Roger Hindley. The completeness theorem for typing lambda-terms. *Theoretical Computer Science*, 22:1–17, 1983.
[33] Daniel Hirschkoff, Jean-Marie Madiot, and Davide Sangiorgi. Duality and i/o-types in the π-calculus. In *CONCUR*, volume 7454 of *LNCS*, pages 302–316. Springer, 2012.
[34] Daniel Hirschkoff, Jean-Marie Madiot, and Davide Sangiorgi. Name-passing calculi: from fusions to preorders and types. In *LICS*, pages 378–387. IEEE Computer Society, 2013.
[35] Kohei Honda, Aybek Mukhamedov, Gary Brown, Tzu-Chun Chen, and Nobuko Yoshida. Scribbling interactions with a formal foundation. In *ICDCIT*, volume 6536 of *LNCS*, pages 55–75. Springer, 2011.
[36] Kohei Honda, Vasco T. Vasconcelos, and Makoto Kubo. Language primitives and type disciplines for structured communication-based programming. In *ESOP*, volume 1381 of *LNCS*, pages 122–138. Springer, 1998.
[37] Kohei Honda and Nobuko Yoshida. On reduction-based process semantics. *Theorical Computer Science*, 151(2):437–486, 1995.
[38] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty Asynchronous Session Types. In *POPL*, pages 273–284. ACM Press, 2008.
[39] Raymond Hu, Dimitrios Kouzapas, Olivier Pernet, Nobuko Yoshida, and Kohei Honda. Type-safe eventful sessions in Java. In *ECOOP*, volume 6183 of *LNCS*, pages 329–353. Springer, 2010.
[40] Raymond Hu, Nobuko Yoshida, and Kohei Honda. Session-based distributed programming in Java. In *ECOOP*, volume 5142 of *LNCS*, pages 516–541. Springer, 2008.
[41] Atsushi Igarashi and Naoki Kobayashi. A generic type system for the pi-calculus. *Theoretical Computer Science*, 311(1-3):121–163, 2004.
[42] Naoki Kobayashi, Benjamin C. Pierce, and David N. Turner. Linearity and the pi-Calculus. *ACM Transactions on Programming Languages and Systems*, 21(5):914–947, 1999.
[43] Dimitrios Kouzapas, Nobuko Yoshida, and Kohei Honda. On asynchronous session semantics. In *FMOODS/FORTE*, volume 6722 of *LNCS*, pages 228–243. Springer, 2011.
[44] Jay Ligatti, Jeremy Blackburn, and Michael Nachtigal. On subtyping-relation completeness, with an application to iso-recursive types. *ACM Transactions on Programming Languages and Systems*, 39(1):4:1–4:36, 2017.
[45] Barbara Liskov and Jeannette M. Wing. A behavioral notion of subtyping. *ACM Transactions on Programming Languages and Systems*, 16(6):1811–1841, 1994.
[46] Robin Milner. The polyadic π-calculus: a tutorial. In *Logic and Algebra of Specification*. Springer, 1991.
[47] Robin Milner and Davide Sangiorgi. Barbed bisimulation. In *ICALP*, volume 623 of *LNCS*, pages 685–695. Springer-Verlag, 1992.
[48] Dimitris Mostrous. *Session Types in Concurrent Calculi: Higher-Order Processes and Objects*. PhD thesis, Imperial College London, 2009.
[49] Dimitris Mostrous and Nobuko Yoshida. Session-based communication optimisation for higher-order mobile processes. In *TLCA*, volume 5608 of *LNCS*, pages 203–218. Springer, 2009.
[50] Dimitris Mostrous and Nobuko Yoshida. Session typing and asynchronous subtyping for the higher-order π-calculus. *Information and Computation*, 241:227 – 263, 2015.
[51] Dimitris Mostrous, Nobuko Yoshida, and Kohei Honda. Global principal typing in partially commutative asynchronous sessions. In *ESOP*, volume 5502 of *LNCS*, pages 316–332. Springer, 2009.
[52] Nicholas Ng, Nobuko Yoshida, and Kohei Honda. Multiparty Session C: Safe parallel programming with message optimisation. In *TOOLS*, volume 7304 of *LNCS*, pages 202–218. Springer, 2012.
[53] Luca Padovani. Fair subtyping for multi-party session types. In *COORDINATION*, volume 6721 of *LNCS*, pages 127–141. Springer, 2011.
[54] Luca Padovani. Fair subtyping for open session types. In *ICALP*, volume 7966 of *LNCS*, pages 373–384. Springer, 2013.
[55] Jorge A. Pérez, Luís Caires, Frank Pfenning, and Bernardo Toninho. Linear logical relations for session-based concurrency. In *ESOP*, volume 7211 of *LNCS*, pages 539–558. Springer, 2012.
[56] Benjamin C. Pierce. *Types and Programming Languages*. MIT Press, 2002.

[57] Benjamin C. Pierce and Davide Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):376–385, 1996.
[58] Benjamin C. Pierce and David N. Turner. Pict: A programming language based on the pi-calculus. In *Proof, Language and Interaction: Essays in Honour of Robin Milner*. MIT Press, 2000.
[59] James Riely and Matthew Hennessy. Trust and partial typing in open systems of mobile agents. In *POPL*, pages 93–104. ACM Press, 1999.
[60] Scribble Project homepage. `http://scribble.github.io/`.
[61] Peter Sewell, Paweł T. Wojciechowski, and Asis Unyapoth. Nomadic Pict: Programming languages, communication infrastructure overlays, and semantics for mobile computation. *ACM Transactions on Programming Languages and Systems*, 32(4):12:1–12:63, April 2010.
[62] Vasco Thudichum Vasconcelos. Fundamentals of session types. In *SFM*, volume 5569 of *LNCS*, pages 158–186. Springer, 2009.
[63] Vasco Thudichum Vasconcelos. Session types for linear multithreaded functional programming. In *PPDP*, pages 1–6. ACM Press, 2009.
[64] Jules Villard. *Heaps and Hops*. PhD thesis, ENS Cachan, 2011.
[65] Phil Wadler. Proposition as sessions. In *ICFP*, pages 273–286. ACM Press, 2012.
[66] Nobuko Yoshida and Vasco Thudichum Vasconcelos. Language primitives and type discipline for structured communication-based programming revisited: Two systems for higher-order session communication. In *SecReT*, volume 171 of *ENTCS*, pages 73–93. Elsevier, 2007.
[67] Nobuko Yoshida, Vasco Thudichum Vasconcelos, Hervé Paulino, and Kohei Honda. Session-based compilation framework for multicore programming. In *FMCO*, volume 5751 of *LNCS*, pages 226–246. Springer, 2009.

Appendix A. Proofs of Section 2

Theorem A.1. *The relation $\leqslant_{\mathsf{s}}$ is transitive.*

Proof. Let $T \leqslant_{\mathsf{s}}^{+} S$ if there exists V such that $T \leqslant_{\mathsf{s}} V$ and $V \leqslant_{\mathsf{s}} S$. It suffices to show $\leqslant_{\mathsf{s}}^{+} \subseteq \leqslant_{\mathsf{s}}$, i.e. that $\leqslant_{\mathsf{s}}^{+}$ satisfies all the rules of Table 6. We consider all the possible shapes of T, V, and S based on the rules of Table 6:

(1) $T = \mathtt{end}$ and $V = \mathtt{end}$ and $S = \mathtt{end}$. Then $T \leqslant_{\mathsf{s}}^{+} S$ agrees with rule [sub-end].
(2) $T = \&_{i\in I\cup J\cup H}?l_i(S_i).T_i$ and $V = \&_{i\in I\cup J}?l_i(S'_i).T'_i$ and $S = \&_{i\in I}?l_i(S''_i).T''_i$ and $S_i \leqslant_{\mathsf{s}} S'_i$ and $T_i \leqslant_{\mathsf{s}} T'_i$ for all $i \in I \cup J$, and $S'_i \leqslant_{\mathsf{s}} S''_i$ and $T'_i \leqslant_{\mathsf{s}} T''_i$ for all $i \in I$. By the definition of $\leqslant_{\mathsf{s}}^{+}$, we get $S_i \leqslant_{\mathsf{s}}^{+} S''_i$ and $T_i \leqslant_{\mathsf{s}}^{+} T''_i$ for all $i \in I$. Then $T \leqslant_{\mathsf{s}}^{+} S$ agrees with rule [sub-bra].
(3) $T = \bigoplus_{i\in I} !l_i\langle S_i\rangle.T_i$ and $V = \bigoplus_{i\in I\cup J} !l_i\langle S'_i\rangle.T'_i$ and $S = \bigoplus_{i\in I\cup J\cup H} !l_i\langle S''_i\rangle.T''_i$ and $S'_i \leqslant_{\mathsf{s}} S_i$ and $T_i \leqslant_{\mathsf{s}} T'_i$ for all $i \in I$, and $S''_i \leqslant_{\mathsf{s}} S'_i$ and $T'_i \leqslant_{\mathsf{s}} T''_i$ for all $i \in I \cup J$. By the definition of $\leqslant_{\mathsf{s}}^{+}$, we get $S''_i \leqslant_{\mathsf{s}}^{+} S_i$ and $T_i \leqslant_{\mathsf{s}}^{+} T''_i$ for all $i \in I$. Then $T \leqslant_{\mathsf{s}}^{+} S$ agrees with rule [sub-sel]. □

The remainder of this section is devoted to the proof of the subject reduction theorem and of the soundness of the synchronous subtyping.

Lemma A.2 (Inversion lemma for synchronous processes)**.**
(1) *If $\Gamma \vdash_{\mathsf{s}} \mathbf{0} \triangleright \Delta$, then Δ is end-only.*
(2) *If $\Gamma \vdash_{\mathsf{s}} X\langle\tilde{u}\rangle \triangleright \Delta$, then $\Gamma = \Gamma', X : \langle\tilde{T}\rangle$ and $\{\tilde{u} : \tilde{T}\} \leqslant_{\mathsf{s}} \Delta$.*
(3) *If $\Gamma \vdash_{\mathsf{s}} \sum_{i\in I} u?l_i(x_i).P_i \triangleright \Delta$, then $\Delta = \Delta', u : \&_{j\in J}?l_j(S_j).T_j$ and $J \subseteq I$ and $\forall j \in J : \Gamma \vdash_{\mathsf{s}} P_j \triangleright \Delta', u : T_j, x_j : S_j$.*
(4) *If $\Gamma \vdash_{\mathsf{s}} u!l\langle u'\rangle.P \triangleright \Delta$, then $\Delta = \Delta', u : T, u' : S$ and $!l\langle S\rangle.T' \leqslant_{\mathsf{s}} T$, and $\Gamma \vdash_{\mathsf{s}} P \triangleright \Delta', u : T'$.*
(5) *If $\Gamma \vdash_{\mathsf{s}} P_1 \mid P_2 \triangleright \Delta$, then $\Delta = \Delta_1, \Delta_2$ and $\Gamma \vdash_{\mathsf{s}} P_1 \triangleright \Delta_1$ and $\Gamma \vdash_{\mathsf{s}} P_2 \triangleright \Delta_2$.*
(6) *If $\Gamma \vdash_{\mathsf{s}} P_1 \oplus P_2 \triangleright \Delta$, then $\Gamma \vdash_{\mathsf{s}} P_1 \triangleright \Delta$ and $\Gamma \vdash_{\mathsf{s}} P_2 \triangleright \Delta$.*

(7) *If* $\Gamma \vdash_{\mathsf{s}} \mathtt{def}\ X(\tilde{y}) = P\ \mathtt{in}\ Q \triangleright \Delta$, *then* $\Gamma, X : \langle \tilde{T} \rangle \vdash_{\mathsf{s}} P \triangleright \{\tilde{y} : \tilde{T}\}$ *and* $\Gamma, X : \langle \tilde{T} \rangle \vdash_{\mathsf{s}} Q \triangleright \Delta$.
(8) *If* $\Gamma \vdash_{\mathsf{s}} (\nu ab)P \triangleright \Delta$, *then* $\Gamma \vdash_{\mathsf{s}} P \triangleright \Delta, a : T, b : \overline{T}$.

Proof. By induction on derivations. □

Lemma A.3 (Substitution lemma for synchronous processes)**.** *If* $\Gamma \vdash_{\mathsf{s}} P \triangleright \Delta, y : T$ *and* $a \notin \mathsf{dom}(\Delta)$, *then* $\Gamma \vdash_{\mathsf{s}} P\{^{a}/_{y}\} \triangleright \Delta, a : T$.

Proof. Standard. □

Lemma A.4. *If* $\Gamma \vdash_{\mathsf{s}} P \triangleright \Delta$ *and* $P \equiv P'$, *then* $\Gamma \vdash_{\mathsf{s}} P' \triangleright \Delta$.

Proof. The proof by induction on $\equiv$ is easy. □

Theorem 2.2. (Subject reduction for synchronous processes) If $\Gamma \vdash_{\mathsf{s}} P \triangleright \Delta$ and $P \to_{\mathsf{s}}^{*} Q$, then $\Gamma \vdash_{\mathsf{s}} Q \triangleright \Delta$.

Proof. We first prove that if $\Gamma \vdash_{\mathsf{s}} P \triangleright \Delta$ and $P \to_{\mathsf{s}} Q$, then $\Gamma \vdash_{\mathsf{s}} Q \triangleright \Delta$. The proof is by induction on the derivation of $P \to_{\mathsf{s}} Q$. We consider some interesting rules of Table 3.

(1) Case [R-COM-SYNC]:

$$\Gamma \vdash_{\mathsf{s}} (\nu ab)(a!l_k\langle c \rangle.P \mid \sum_{i \in I} b?l_i(x_i).Q_i) \triangleright \Delta \tag{A.1}$$

where $k \in I$. By applying Lemma A.2.8 to (A.1), we have

$$\Gamma \vdash_{\mathsf{s}} a!l_k\langle c \rangle.P \mid \sum_{i \in I} b?l_i(x_i).Q_i \triangleright \Delta, a : T, b : \overline{T} \tag{A.2}$$

By applying Lemma A.2.5 to (A.2), we get

$$\Gamma \vdash_{\mathsf{s}} a!l_k\langle c \rangle.P \triangleright \Delta_1 \tag{A.3}$$

$$\Gamma \vdash_{\mathsf{s}} \sum_{i \in I} b?l_i(x_i).Q_i \triangleright \Delta_2 \tag{A.4}$$

where $\Delta_1, \Delta_2 = \Delta, a : T, b : \overline{T}$. By applying Lemma A.2.4 to (A.3), we have

$$\Delta_1 = \Delta_1', a : T, c : S$$

$$!l_k\langle S \rangle.T' \leqslant_{\mathsf{s}} T \tag{A.5}$$

$$\Gamma \vdash_{\mathsf{s}} P \triangleright \Delta_1', a : T' \tag{A.6}$$

By applying Lemma A.2.3 to (A.4), we have

$$\Delta_2 = \Delta_2', b : \overline{T}$$

$$\overline{T} = \&_{j \in J} ?l_j(S_j).T_j, \quad J \subseteq I \tag{A.7}$$

$$\forall j \in J : \Gamma \vdash_{\mathsf{s}} Q_j \triangleright \Delta_2', b : T_j, x_j : S_j \tag{A.8}$$

By (A.7) and duality,

$$T = \overline{\&_{j \in J} ?l_j(S_j).T_j} = \bigoplus_{j \in J} !l_j\langle S_j \rangle.\overline{T_j} \tag{A.9}$$

Then by (A.5) $k \in J$ and

$$S = S_k \tag{A.10}$$

which together with (A.8) imply

$$\Gamma \vdash_{\mathsf{s}} Q_k \triangleright \Delta_2', b : T_k, x_k : S_k \tag{A.11}$$

By (A.10) and by applying Lemma A.3 to (A.11), we have

$$\Gamma \vdash_{\mathsf{s}} Q_k\{^c/_{x_k}\} \triangleright \Delta_2', b : T_k, c : S \tag{A.12}$$

From (A.5) and (A.9), we get

$$T' \leqslant_{\mathsf{s}} \overline{T}_k \tag{A.13}$$

Applying [T-SUB] to (A.6) and (A.13), we derive

$$\Gamma \vdash_{\mathsf{s}} P \triangleright \Delta_1', a : \overline{T}_k \tag{A.14}$$

By applying [T-PAR] to (A.12) and (A.14), we derive

$$\Gamma \vdash_{\mathsf{s}} P \mid Q_k\{^c/_{x_k}\} \triangleright \Delta_1', \Delta_2', a : \overline{T}_k, b : T_k, c : S \tag{A.15}$$

By applying [T-NEW-SYNC] to (A.15), we derive $\Gamma \vdash_{\mathsf{s}} (\nu ab)(P \mid Q_k\{^c/_{x_k}\}) \triangleright \Delta_1', \Delta_2', c : S$. Since $\Delta_1', \Delta_2', c : S = \Delta$, we conclude

$$\Gamma \vdash_{\mathsf{s}} (\nu ab)(P \mid Q_k\{^c/_{x_k}\}) \triangleright \Delta$$

(2) Case [R-DEF]:

$$\Gamma \vdash_{\mathsf{s}} \mathtt{def}\ X(\tilde{y}) = P\ \mathtt{in}\ (X\langle \tilde{a} \rangle \mid Q) \triangleright \Delta \tag{A.16}$$

Applying Lemma A.2.7 to (A.16), we have

$$\Gamma, X : \langle \tilde{T} \rangle \vdash_{\mathsf{s}} P \triangleright \{\tilde{y} : \tilde{T}\} \tag{A.17}$$

$$\Gamma, X : \langle \tilde{T} \rangle \vdash_{\mathsf{s}} X\langle \tilde{a} \rangle \mid Q \triangleright \Delta \tag{A.18}$$

Applying Lemma A.2.5 to (A.18), we have

$$\Gamma, X : \langle \tilde{T} \rangle \vdash_{\mathsf{s}} X\langle \tilde{a} \rangle \triangleright \Delta_1 \tag{A.19}$$

$$\Gamma, X : \langle \tilde{T} \rangle \vdash_{\mathsf{s}} Q \triangleright \Delta_2 \tag{A.20}$$

where $\Delta = \Delta_1, \Delta_2$. By Lemma A.2.2 and (A.19), we have

$$\{\tilde{a} : \tilde{T}\} \leqslant_{\mathsf{s}} \Delta_1 \tag{A.21}$$

Applying Lemma A.3 to (A.17), we get

$$\Gamma, X : \langle \tilde{T} \rangle \vdash_{\mathsf{s}} P\{^{\tilde{a}}/_{\tilde{y}}\} \triangleright \{\tilde{a} : \tilde{T}\} \tag{A.22}$$

By applying [T-SUB] to (A.21) and (A.22), we derive

$$\Gamma, X : \langle \tilde{T} \rangle \vdash_{\mathsf{s}} P\{^{\tilde{a}}/_{\tilde{y}}\} \triangleright \Delta_1 \tag{A.23}$$

By applying [T-PAR] to (A.20) and (A.23), we derive

$$\Gamma, X : \langle \tilde{T} \rangle \vdash_{\mathsf{s}} P\{^{\tilde{a}}/_{\tilde{y}}\} \mid Q \triangleright \Delta_1, \Delta_2 \tag{A.24}$$

By applying [T-DEF] to (A.24), we conclude

$$\Gamma \vdash_{\mathsf{s}} \mathtt{def}\ X(\tilde{y}) = P\ \mathtt{in}\ (P\{^{\tilde{a}}/_{\tilde{y}}\} \mid Q) \triangleright \Delta_1, \Delta_2$$

where $\Delta_1, \Delta_2 = \Delta$.

(3) Case [R-CONTEXT]: We consider the context $(\nu ab)C$. The proofs for other cases are similar and simpler. Let

$$\Gamma \vdash_{\mathsf{s}} (\nu ab)C[P] \triangleright \Delta \tag{A.25}$$

By applying A.2.8 to (A.25), we have

$$\Gamma \vdash_{\mathsf{s}} C[P] \triangleright \Delta, a : T, b : \overline{T}$$

By induction, we derive

$$\Gamma \vdash_{\mathsf{s}} C[Q] \triangleright \Delta, a : T, b : \overline{T} \tag{A.26}$$

By applying [T-NEW-SYNC] to (A.26), we conclude

$$\Gamma \vdash_{\mathsf{s}} (\nu ab)C[Q] \triangleright \Delta$$

We have proved that if $\Gamma \vdash_{\mathsf{s}} P \triangleright \Delta$ and $P \to_{\mathsf{s}} Q$, then $\Gamma \vdash_{\mathsf{s}} Q \triangleright \Delta$. Now, the main statement of the theorem can be easily proved by induction on the length of the sequence of transitions $P \to_{\mathsf{s}}^* Q$. □

As a consequence of subject reduction we get a substitution lemma for process variables, which can be also proved independently by induction on reduction contexts.

Lemma A.5. *If $\Gamma, X : \langle T \rangle \vdash_{\mathsf{s}} C[X\langle a \rangle] \triangleright \Delta$ and $\Gamma \vdash_{\mathsf{s}} P \triangleright \{a : T\}$ and X does not occur free in C, then $\Gamma \vdash_{\mathsf{s}} C[P] \triangleright \Delta$.*

Proof. The proof is by structural induction on reduction contexts.

(1) If C is empty, i.e. $C = [\ ]$, by Lemma A.2.2, $\{a : T\} \leqslant_{\mathsf{s}} \Delta$. By [T-SUB], we derive $\Gamma \vdash_{\mathsf{s}} P \triangleright \Delta$.

(2) If $C = (\nu cd)C'$, let

$$\Gamma, X : \langle T \rangle \vdash_{\mathsf{s}} (\nu cd)C'[X\langle a \rangle] \triangleright \Delta \tag{A.27}$$

$$\Gamma \vdash_{\mathsf{s}} P \triangleright \{a : T\} \tag{A.28}$$

By applying Lemma A.2.8 to (A.27), we have

$$\Gamma, X : \langle T \rangle \vdash_{\mathsf{s}} C'[X\langle a \rangle] \triangleright \Delta, c : T, d : \overline{T} \tag{A.29}$$

By induction, (A.28) and (A.29) together imply

$$\Gamma \vdash_{\mathsf{s}} C'[P] \triangleright \Delta, c : T, d : \overline{T} \tag{A.30}$$

By applying [T-NEW-SYNC] to (A.30), we derive

$$\Gamma \vdash_{\mathsf{s}} (\nu cd)C'[P] \triangleright \Delta$$

Notice that this proof holds also if $a = c$ or $a = d$.

(3) If $C = C' \mid Q$, let

$$\Gamma, X : \langle T \rangle \vdash_{\mathsf{s}} C'[X\langle a \rangle] \mid Q \triangleright \Delta \tag{A.31}$$

and (A.28). By applying Lemma A.2.5 to (A.31) and by the assumption that X does not occur free in Q, we have

$$\Gamma, X : \langle T \rangle \vdash_{\mathsf{s}} C'[X\langle a \rangle] \triangleright \Delta_1 \tag{A.32}$$

$$\Gamma \vdash_{\mathsf{s}} Q \triangleright \Delta_2 \tag{A.33}$$

where $\Delta_1, \Delta_2 = \Delta$. By induction, (A.28) and (A.32) together imply

$$\Gamma \vdash_{\mathsf{s}} C'[P] \triangleright \Delta_1 \tag{A.34}$$

By applying [T-PAR] to (A.33) and (A.34), we derive

$$\Gamma \vdash_{\mathsf{s}} C'[P] \mid Q \triangleright \Delta$$

(4) If $C = \mathtt{def}\ D\ \mathtt{in}\ C'$, let

$$\Gamma, X : \langle \tilde{T} \rangle \vdash_{\mathsf{s}} \mathtt{def}\ D\ \mathtt{in}\ C'[X\langle \tilde{a} \rangle] \triangleright \Delta \tag{A.35}$$

where $D = (Y(\tilde{y}) = P')$ and $Y \neq X$, and (A.28). By applying Lemma A.2.7 to (A.35) and by the assumption that X does not occur free in D, we have

$$\Gamma, Y : \langle \tilde{T'} \rangle \vdash_{\mathsf{s}} P' \triangleright \{\tilde{y} : \tilde{T'}\} \tag{A.36}$$

$$\Gamma, X : \langle T \rangle, Y : \langle \tilde{T'} \rangle \vdash_{\mathsf{s}} C'[X\langle \tilde{a} \rangle] \triangleright \Delta \tag{A.37}$$

By induction, (A.28) and (A.37) imply

$$\Gamma, Y : \langle \tilde{T'} \rangle \vdash_{\mathsf{s}} C'[P] \triangleright \Delta \tag{A.38}$$

By applying [T-DEF] to (A.36) and (A.38), we derive

$$\Gamma \vdash_{\mathsf{s}} \mathtt{def}\ Y(\tilde{y}) = P'\ \mathtt{in}\ C'[P] \triangleright \Delta$$

□

Theorem 2.4. The synchronous subtyping relation is sound for the synchronous calculus.

Proof. Suppose $T \leqslant_{\mathsf{s}} S$ and $\Gamma \vdash_{\mathsf{s}} P \triangleright \{a : T\}$. By applying [T-SUB] we have $\Gamma \vdash_{\mathsf{s}} P \triangleright \{a : S\}$. By definition of $C[a : S]$, we have $\Gamma, X : \langle S \rangle \vdash_{\mathsf{s}} C[X\langle a \rangle] \triangleright \Delta$. By Lemma A.5, we can get $\Gamma \vdash_{\mathsf{s}} C[P] \triangleright \Delta$. By Corollary 2.3, we have $C[P] \not\rightarrow_{\mathsf{s}}^{*} \mathtt{error}$. □

Appendix B. Proofs of Section 4

Proposition B.1. $\& \in \mu \mathbf{t}.T$ if and only if $\& \in T\{^{\mu \mathbf{t}.T}/_{\mathbf{t}}\}$.

Proof. ($\Longrightarrow$). Assume $\& \in \mu \mathbf{t}.T$, for some derivation $\mathcal{D}$. We observe that $\mathcal{D}$ can only conclude by the rule for recursion on page 22, and from its premise we have a derivation $\mathcal{D}'$ proving $\& \in T$. We observe that $\mathcal{D}'$ cannot reach $\mathbf{t}$ in T, where no rule is defined: hence, if we inductively rewrite $\mathcal{D}'$ substituting each occurrence of $\mathbf{t}$ with $\mu \mathbf{t}.T$, we obtain a derivation proving $\& \in T\{^{\mu \mathbf{t}.T}/_{\mathbf{t}}\}$.

($\Longleftarrow$). Assuming $\& \in T\{^{\mu \mathbf{t}.T}/_{\mathbf{t}}\}$, for some derivation $\mathcal{D}$, we have two cases. If $\mathcal{D}$ traverses a substitution of $\mathbf{t}$ in T with $\mu \mathbf{t}.T$, then we have a corresponding sub-derivation $\mathcal{D}'$ proving $\& \in \mu \mathbf{t}.T$, which is the thesis[2]. Otherwise, $\mathcal{D}$ reaches its axioms *without* traversing any substitution of $\mathbf{t}$ in T with $\mu \mathbf{t}.T$: hence, by inductively rewriting $\mathcal{D}$ removing such a substitution and restoring $\mathbf{t}$ in all terms, we obtain a derivation $\mathcal{D}'$ proving $\& \in T$. By using $\mathcal{D}'$ as a premise for the rule for recursion on page 22, we obtain a derivation proving $\& \in \mu \mathbf{t}.T$. □

[2]We could also prove that this case is actually absurd, but this detail is not relevant for the main proof.

To show the transitivity of asynchronous subtyping (Theorem 4.8) we extend the asynchronous subtyping to asynchronous contexts (Table 24). It is easy to verify that if $\mathcal{A} \leqslant_{\mathsf{a}} \mathcal{A}'$, and N and N' are the set of hole indices of $\mathcal{A}$ and $\mathcal{A}'$, respectively, then $N' \subseteq N$. If previous conditions hold and $T_n \leqslant_{\mathsf{a}} T'_n$ for all $n \in N'$, then $\mathcal{A}[T_n]^{n\in N} \leqslant_{\mathsf{a}} \mathcal{A}'[T'_n]^{n\in N'}$. Two more

$$[\text{SUB-EMPTY}]\quad [\,]^n \leqslant [\,]^n \qquad [\text{SUB-CONT}]\quad \frac{\forall i \in I : S_i \leqslant S'_i \quad \mathcal{A}_i \leqslant \mathcal{A}'_i}{\&_{i\in I\cup J}?l_i(S_i).\mathcal{A}_i \leqslant \&_{i\in I}?l_i(S'_i).\mathcal{A}'_i}$$

Table 24: Subtyping for asynchronous contexts.

lemmas on properties of $\leqslant_{\mathsf{a}}$ are handy.

Lemma B.2. *If $T \leqslant_{\mathsf{a}} S$ and $\& \in S$, then $\& \in T$.*

Proof. By cases on the rules defining $\leqslant_{\mathsf{a}}$. □

Lemma B.3. *If $\mathcal{A}[\bigoplus_{i\in I_n} !l_i^n\langle S_i^n\rangle.T_i^n]^{n\in N} \leqslant_{\mathsf{a}} T$, then there exists $\mathcal{A}'[\,]^{n\in N'}$ such that:*
(1) *$\mathcal{A} \leqslant_{\mathsf{a}} \mathcal{A}'$ and $N' \subseteq N$*
(2) *for all $n \in N'$, there exists $\mathcal{A}_n[\,]^{m\in M_n}$ such that*
- *for all $i \in I_n$ and $m \in M_n$, there exist $H_{n,m}$, $S_i^{n,m}$ and $T_i^{n,m}$ such that:*
  (a) $S_i^{n,m} \leqslant_{\mathsf{a}} S_i^n$
  (b) $T_i^n \leqslant_{\mathsf{a}} \mathcal{A}_n[T_i^{n,m}]^{m\in M_n}$
  (c) $T = \mathcal{A}'[\mathcal{A}_n[\bigoplus_{i\in I_n\cup H_{n,m}} !l_i^n\langle S_i^{n,m}\rangle.T_i^{n,m}]^{m\in M_n}]^{n\in N'}$

Proof. By structural induction on $\mathcal{A}$.

In the base case where $\mathcal{A}$ is just one hole (i.e., N is a singleton), then we choose $\mathcal{A}'$ to be just one hole, too; hence, we get $\mathcal{A} \leqslant_{\mathsf{a}} \mathcal{A}'$ (by rule [SUB-EMPTY]) and $N' = N = \{1\}$, thus satisfying item 1 of the statement. Then, according to the rules of Table 6, we have either:
- by rule [SUB-SEL], $T = \bigoplus_{i\in I_1\cup J} !l_i\langle S_i^1\rangle.T_i^1$ and $S_i^1 \leqslant_{\mathsf{a}} S_i$ and $T_i \leqslant_{\mathsf{a}} T_i^1$ for all $i \in I_1$. Then we let:
  – $\mathcal{A}_1$ be just one hole (i.e., $M_1 = \{1\}$ is a singleton);
  – $H_{1,1} = J$, and for all $i \in I_1$, $S_i^{1,1} = S_i^1$ and $T_i^{1,1} = T_i^1$;
- otherwise, by rule [SUB-PERM-ASYNC], there exists $\mathcal{A}''[\,]^{m\in M}$ such that $\& \in \mathcal{A}''$ and $T = \mathcal{A}''[\bigoplus_{i\in I\cup J_m} !l_i\langle S_i^m\rangle.T_i^m]^{m\in M}$, and for all $i \in I_1$, $T_i \leqslant_{\mathsf{a}} \mathcal{A}''[T_i^m]^{m\in M}$ and for all $m \in M$, $S_i^m \leqslant_{\mathsf{a}} S_i$. Then we let:
  – $\mathcal{A}_1 = \mathcal{A}''$ (and thus, $M_1 = M$);
  – for all $m \in M_1$, $H_{1,m} = J_m$, and for all $i \in I_1$, $S_i^{1,m} = S_i^m$ and $T_i^{1,m} = T_i^m$.

In both cases, we satisfy items 2a, 2b and 2c of the statement.

For the induction step, let $\mathcal{A} = \&_{j\in J}?l_j(S_j).\mathcal{A}_j$ with N_j being the set of hole indices occurring in $\mathcal{A}_j$ and $N = \bigcup_{j\in J} N_j$. Since by assumption

$$\&_{j\in J}?l_j(S_j).\mathcal{A}_j[\textstyle\bigoplus_{i\in I_n} !l_i^n\langle S_i^n\rangle.T_i^n]^{n\in N_j} \leqslant_{\mathsf{a}} T$$

and since such a relation can only hold by rule [SUB-BRA], the shape of T must be

$$T = \&_{j\in J'}?l_j(S'_j).T'_j \tag{B.1}$$

where $J' \subseteq J$ and, for all $j \in J'$, $S_j \leqslant_{\mathsf{a}} S'_j$ and $\mathcal{A}_j[\bigoplus_{i\in I_n} !l_i^n\langle S_i^n\rangle.T_i^n]^{n\in N_j} \leqslant_{\mathsf{a}} T'_j$. By the induction hypothesis, for all $j \in J'$, there exists $\mathcal{A}'_j[\,]^{n\in N'_j}$ such that (by item 2c):

$$T'_j \;=\; \mathcal{A}'_j[\mathcal{A}_n[\textstyle\bigoplus_{i\in I_n\cup H_{n,m}} !l_i^n\langle S_i^{n,m}\rangle.T_i^{n,m}]^{m\in M_n}]^{n\in N'_j} \tag{B.2}$$

where:

- $\mathcal{A}_j \leqslant_{\mathsf{a}} \mathcal{A}'_j$ and $N'_j \subseteq N_j$ (from item 1 of the statement);
- for all $n \in N'_j$, $i \in I_n$ and $m \in M_n$, we have:
  (1) $S_i^{n,m} \leqslant_{\mathsf{a}} S_i^n$ (from item 2a), and
  (2) $T_i^n \leqslant_{\mathsf{a}} \mathcal{A}_n[T_i^{n,m}]^{m\in M_n}$ (from item 2b).

Now, let $\mathcal{A}' = \&_{j\in J'}?l_j(S'_j).\mathcal{A}'_j$ and $N' = \bigcup_{j\in J'} N'_j$. Since for all $j \in J' \subseteq J$ we have $N'_j \subseteq N_j$, we also get $N' \subseteq N$; moreover, since for all $j \in J'$ we have $S_j \leqslant_{\mathsf{a}} S'_j$ and $\mathcal{A}_j \leqslant_{\mathsf{a}} \mathcal{A}'_j$, by [sub-cont] we also get $\mathcal{A} \leqslant_{\mathsf{a}} \mathcal{A}'$: hence, we satisfy item 1 of the statement. Furthermore, from items 1 and 2 above we satisfy respectively items 2a and 2b of the statement. Finally, from (B.1) and (B.2) we obtain:

$$T \;=\; \&_{j\in J'}?l_j(S'_j).\mathcal{A}'_j[\mathcal{A}_n[\textstyle\bigoplus_{i\in I_n\cup H_{n,m}} !l_i^n\langle S_i^{n,m}\rangle.T_i^{n,m}]^{m\in M_n}]^{n\in N'_j}$$

from which we get:

$$T \;=\; \mathcal{A}'[\mathcal{A}_n[\textstyle\bigoplus_{i\in I_n\cup H_{n,m}} !l_i^n\langle S_i^{n,m}\rangle.T_i^{n,m}]^{m\in M_n}]^{n\in N'}$$

thus satisfying item 2c of the statement. □

Theorem 4.8. The relation $\leqslant_{\mathsf{a}}$ is transitive.

Proof. It suffices to show $\leqslant_{\mathsf{a}}^+ \subseteq \leqslant_{\mathsf{a}}$, where $T \leqslant_{\mathsf{a}}^+ S$ if there exists V such that $T \leqslant_{\mathsf{a}} V$ and $V \leqslant_{\mathsf{a}} S$. We proceed by cases on the rules concluding $T \leqslant_{\mathsf{a}} V$ and $V \leqslant_{\mathsf{a}} S$.

If $T \leqslant_{\mathsf{a}} V$ by rule [sub-sel] (respectively [sub-bra]), and $V \leqslant_{\mathsf{a}} S$ by rule [sub-sel] (respectively [sub-bra]), then the proof is straightforward, and $T \leqslant_{\mathsf{a}} S$ holds again by rule [sub-sel] (respectively [sub-bra]).

If $T \leqslant_{\mathsf{a}} V$ by rule [sub-sel] (respectively [sub-perm-async]), and $V \leqslant_{\mathsf{a}} S$ by rule [sub-perm-async] (respectively [sub-bra]), then we have $T = \bigoplus_{i\in I} !l_i\langle S_i\rangle.T_i$ and $V = \mathcal{A}[\bigoplus_{i\in I\cup J_n} !l_i\langle S_i^n\rangle.T_i^n]^{n\in N}$ (where $\mathcal{A}$ is just one hole when $T \leqslant_{\mathsf{a}} V$ holds by [sub-sel]). From $V \leqslant_{\mathsf{a}} S$, by Lemma B.3 (item 2c) we have:

$$S \;=\; \mathcal{A}'[\mathcal{A}_n[\textstyle\bigoplus_{i\in I\cup J_n\cup H_{n,m}} !l_i\langle S_i^{n,m}\rangle.T_i^{n,m}]^{m\in M_n}]^{n\in N'}$$

where:

(1) $\mathcal{A} \leqslant_{\mathsf{a}} \mathcal{A}'$ and $N' \subseteq N$ (by item 1 of Lemma B.3);
(2) for all $n \in N'$, $m \in M_n$ and $i \in I \cup J_n$, we have $S_i^{n,m} \leqslant_{\mathsf{a}} S_i^n$ and $T_i^n \leqslant_{\mathsf{a}} \mathcal{A}_n[T_i^{n,m}]^{m\in M_n}$ (resp. from items 2a and 2b of Lemma B.3).

Moreover, from $T \leqslant_{\mathsf{a}} V$, we also have:

(3) for all $n \in N$, $S_i^n \leqslant_{\mathsf{a}} S_i$;
(4) for all $i \in I$, $T_i \leqslant_{\mathsf{a}} \mathcal{A}[T_i^n]^{n\in N}$.

By the definition of $\leqslant_{\mathsf{a}}^+$, from items 2 and 3 above, we have $S_i^{n,m} \leqslant_{\mathsf{a}}^+ S_i$ for all $i \in I$, $m \in M_n$ and $n \in N'$. Furthermore, from $\mathcal{A} \leqslant_{\mathsf{a}} \mathcal{A}'$ (item 1) and $T_i^n \leqslant_{\mathsf{a}} \mathcal{A}_n[T_i^{n,m}]^{m\in M_n}$ (item 2), for all $i \in I \cup J_n$ and $n \in N'$ we have:

$$\mathcal{A}[T_i^n]^{n\in N} \;\leqslant_{\mathsf{a}}\; \mathcal{A}'[\mathcal{A}_n[T_i^{n,m}]^{m\in M_n}]^{n\in N'} \tag{B.3}$$

Now, according to the definition of $\leqslant_{\mathsf{a}}^{+}$, (B.3) and $T_i \leqslant_{\mathsf{a}} \mathcal{A}[T_i^n]^{n\in N}$ (from item 4) imply $T_i \leqslant_{\mathsf{a}}^{+} \mathcal{A}'[\mathcal{A}_n[T_i^{n,m}]^{m\in M_n}]^{n\in N'}$, for all $i \in I$. Moreover, we observe that since $V \leqslant_{\mathsf{a}} S$ holds by rule [sub-perm-async] (resp. [sub-bra]), then we have $\& \in S$; and since $V \leqslant_{\mathsf{a}} S$ and $T \leqslant_{\mathsf{a}} V$, by applying Lemma B.2 twice we obtain $\& \in V$ and $\& \in T$, i.e. $\& \in T_i$ for all $i \in I$. Thus $T \leqslant_{\mathsf{a}}^{+} S$ agrees with rule [sub-perm-async]. □

The remaining of this section is devoted to the proof of the subject reduction theorem and of the soundness of the asynchronous subtyping.

Lemma B.4 (Inversion lemma for asynchronous processes)**.**

(1) *If* $\Gamma \vdash_{\mathsf{a}} \mathbf{0} \triangleright \Delta$, *then* Δ *is end-only.*
(2) *If* $\Gamma \vdash_{\mathsf{a}} X\langle\tilde{u}\rangle \triangleright \Delta$, *then* $\Gamma = \Gamma', X : \langle\tilde{T}\rangle$ *and* $\{\tilde{u} : \tilde{T}\} \leqslant_{\mathsf{a}} \Delta$.
(3) *If* $\Gamma \vdash_{\mathsf{a}} \sum_{i\in I} u?l_i(x_i).P_i \triangleright \Delta$, *then* $\Delta = \Delta', u : \&_{j\in J} ?l_j(S_j).T_j$ *and* $J \subseteq I$ *and* $\forall j \in J : \Gamma \vdash_{\mathsf{a}} P_j \triangleright \Delta', u : T_j, x_j : S_j$.
(4) *If* $\Gamma \vdash_{\mathsf{a}} u!l\langle u'\rangle.P \triangleright \Delta$, *then* $\Delta = \Delta', u : T, u' : S$ *and* $!l\langle S\rangle.T' \leqslant_{\mathsf{a}} T$ *and* $\Gamma \vdash_{\mathsf{a}} P \triangleright \Delta', u : T'$.
(5) *If* $\Gamma \vdash_{\mathsf{a}} P_1 \mid P_2 \triangleright \Delta$, *then* $\Delta = \Delta_1, \Delta_2$ *and* $\Gamma \vdash_{\mathsf{a}} P_1 \triangleright \Delta_1$ *and* $\Gamma \vdash_{\mathsf{a}} P_2 \triangleright \Delta_2$.
(6) *If* $\Gamma \vdash_{\mathsf{a}} P_1 \oplus P_2 \triangleright \Delta$, *then* $\Gamma \vdash_{\mathsf{a}} P_1 \triangleright \Delta$ *and* $\Gamma \vdash_{\mathsf{a}} P_2 \triangleright \Delta$.
(7) *If* $\Gamma \vdash_{\mathsf{a}} \mathtt{def}\ X(\tilde{y}) = P\ \mathtt{in}\ Q \triangleright \Delta$, *then* $\Gamma, X : \langle\tilde{T}\rangle \vdash_{\mathsf{a}} P \triangleright \{\tilde{y} : \tilde{T}\}$ *and* $\Gamma, X : \langle\tilde{T}\rangle \vdash_{\mathsf{a}} Q \triangleright \Delta$.
(8) *If* $\Gamma \vdash_{\mathsf{a}} (\nu ab)P \triangleright \Delta$, *then* $\Gamma \vdash_{\mathsf{a}} P \triangleright \Delta, a : T_1, b : T_2, ba : \tau_1, ab : \tau_2$ *and* $T_1 - \tau_1 \bowtie T_2 - \tau_2$.
(9) *If* $\Gamma \vdash_{\mathsf{a}} ba \blacktriangleright \varnothing \triangleright \Delta$, *then* $\{ba : \epsilon\} \leqslant_{\mathsf{a}} \Delta$.
(10) *If* $\Gamma \vdash_{\mathsf{a}} ba \blacktriangleright h \cdot l\langle c\rangle \triangleright \Delta$, *then* $\Delta = \Delta', c : S, ba : \tau \cdot l\langle S'\rangle$ *and* $S' \leqslant_{\mathsf{a}} S$ *and* $\Gamma \vdash_{\mathsf{a}} ba \blacktriangleright h \triangleright \Delta', ba : \tau$.

Proof. By induction on derivations. □

Lemma B.5 (Substitution lemma for asynchronous processes)**.** *If* $\Gamma \vdash_{\mathsf{a}} P \triangleright \Delta, y : T$ *and* $a \notin \mathsf{dom}(\Delta)$, *then* $\Gamma \vdash_{\mathsf{a}} P\{^{a}/_{y}\} \triangleright \Delta, a : T$.

Proof. The proof is by induction on the derivation of $\Gamma \vdash_{\mathsf{a}} P \triangleright \Delta, y : T$. The only interesting case is:

$$\frac{\Gamma \vdash_{\mathsf{a}} P \triangleright \Delta, y : T, b : T_1, c : T_2, cb : \tau_1, bc : \tau_2 \quad T_1 - \tau_1 \bowtie T_2 - \tau_2}{\Gamma \vdash_{\mathsf{a}} (\nu bc)P \triangleright \Delta, y : T}$$

By induction $\Gamma \vdash_{\mathsf{a}} P\{^{a}/_{y}\} \triangleright \Delta, a : T, b : T_1, c : T_2, cb : \tau_1, bc : \tau_2$. Thus by [t-new-async], we conclude $\Gamma \vdash_{\mathsf{a}} (\nu bc)P\{^{a}/_{y}\} \triangleright \Delta, a : T$. □

Lemma B.6 (Types of queues)**.** *If* $\Gamma \vdash_{\mathsf{a}} ba \blacktriangleright l\langle c\rangle \cdot h \triangleright \Delta$, *then* $\Delta = \Delta', c : S, ba : l\langle S'\rangle \cdot \tau$, *and* $S' \leqslant_{\mathsf{a}} S$ *and* $\Gamma \vdash_{\mathsf{a}} ba \blacktriangleright h \triangleright \Delta', ba : \tau$.

Proof. By induction on n we show:
If $\Gamma \vdash_{\mathsf{a}} ba \blacktriangleright l_1\langle c_1\rangle \cdot \ldots \cdot l_n\langle c_n\rangle \triangleright \Delta$, then $\Delta = \Delta', c_1 : S_1, \ldots, c_n : S_n, ba : l_1\langle S_1'\rangle \cdot \ldots \cdot l_n\langle S_n'\rangle$, where Δ' is end-only and $S_i' \leqslant_{\mathsf{a}} S_i$ for $1 \leq i \leq n$. The first step follows from Lemma B.4.9. The induction step follows from Lemma B.4.10. □

Theorem B.7. *If* $\Gamma \vdash_{\mathsf{a}} P \triangleright \Delta$ *and* $P \equiv P'$, *then* $\Gamma \vdash_{\mathsf{a}} P' \triangleright \Delta$.

Proof. The proof is by induction on $\equiv$. The most interesting case is rule [s-queue-equiv]. Let $h \equiv h'$ and $\Gamma \vdash_{\mathsf{a}} ab \blacktriangleright h \triangleright \Delta$. The equivalence $h \equiv h'$ should come from one of the following cases: $h \equiv \varnothing \cdot h = h'$, or $h \equiv h \cdot \varnothing = h'$, or $h = h_1 \cdot (h_2 \cdot h_3) \equiv (h_1 \cdot h_2) \cdot h_3 = h'$. For all cases, the messages in h and h' are the same and they are in the same order. Therefore, by Lemma B.6, if $\Gamma \vdash_{\mathsf{a}} ab \blacktriangleright h' \triangleright \Delta'$, then $\Delta' = \Delta$. □

We extend the session remainder to type contexts in the expected way.

Lemma B.8.

$$\mathcal{A}[T_n]^{n\in N} - \tau = \begin{cases} \mathcal{A}'[T_n]^{n\in N'} & \text{if } \mathcal{A}[\,]^{n\in N} - \tau = \mathcal{A}'[\,]^{n\in N'}, \\ T_{n_0} - \tau_0 & \text{if } \mathcal{A}[\,]^{n\in N} - \tau = [\,]^{n_0} - \tau_0. \end{cases}$$

Proof. The proof by cases is easy.

- If $\mathcal{A}[\,]^{n\in N} - \tau = \mathcal{A}'[\,]^{n\in N'}$, we have $\mathcal{A}[T_n]^{n\in N} - \tau = \mathcal{A}'[T_n]^{n\in N'}$.
- If $\mathcal{A}[\,]^{n\in N} - \tau = [\,]^{n_0} - \tau_0$, we have $\mathcal{A}[T_n]^{n\in N} - \tau = T_{n_0} - \tau_0$. □

Lemma 4.12. *If $\Delta \Rightarrow \Delta'$ and Δ is balanced, then Δ' is balanced.*

Proof. By cases on the definition of $\Rightarrow$.

(1) Let $\Delta, a : T, ba : \tau \Rightarrow \Delta', a : T', ba : \tau'$ and $T - \tau$ be defined.
   (a) If the applied rule is [TR-OUT]: $T = \mathcal{A}[\bigoplus_{i\in I_n} !l_i^n\langle S_i^n\rangle.T_i^n]^{n\in N}$ and $\forall n \in N\ \exists i_n \in I_n : l_{i_n}^n = l$ and $S_{i_n}^n \leqslant_{\mathsf{a}} S$, then we get $T' = \mathcal{A}[T_{i_n}^n]^{n\in N}$ and $\tau' = \tau$. By Lemma B.8 we have the following subcases:
      (i) If $\mathcal{A}[\,]^{n\in N} - \tau = \mathcal{A}'[\,]^{n\in N'}$, then $T' - \tau' = \mathcal{A}[T_{i_n}^n]^{n\in N} - \tau = \mathcal{A}'[T_{i_n}^n]^{n\in N'}$, which is defined.
      (ii) If $\mathcal{A}[\,]^{n\in N} - \tau = [\,]^{n_0} - \tau_0$, then $T' - \tau' = \mathcal{A}[T_{i_n}^n]^{n\in N} - \tau = T_{i_{n_0}}^{n_0} - \tau_0$, which is defined because we know that $\bigoplus_{i\in I_{n_0}} !l_i^{n_0}\langle S_i^{n_0}\rangle.T_i^{n_0} - \tau_0$ is defined and, therefore, $\forall i \in I_{n_0} : T_i^{n_0} - \tau_0$ is defined.
   (b) If the applied rule is [TR-IN]: $T = \&_{i\in I}?l_i(S_i).T_i$ and $\tau = l_k\langle S\rangle \cdot \tau'$ and $S_k \leqslant_{\mathsf{a}} S$, and $T' = T_k$ and $k \in I$. We get $\&_{i\in I}?l_i(S_i).T_i - l_k\langle S\rangle \cdot \tau' = T_k - \tau'$. Therefore $T_k - \tau'$ is defined because it is equal to $T - l_k\langle S\rangle \cdot \tau'$, which is defined by assumption.

(2) Let $\Delta, a : T_1, b : T_2, ba : \tau_1, ab : \tau_2 \Rightarrow \Delta', a : T_1', b : T_2', ba : \tau_1', ab : \tau_2'$ and $T_1 - \tau_1 \bowtie T_2 - \tau_2$.
   (a) If the applied rule is [TR-OUT]: $T_1 = \mathcal{A}[\bigoplus_{i\in I_n} !l_i^n\langle S_i^n\rangle.T_i^n]^{n\in N}$ and $\forall n \in N\ \exists k_n \in I_n : l_{k_n}^n = l$ and $S_{k_n}^n \leqslant_{\mathsf{a}} S$ and $T_1' = \mathcal{A}[T_{k_n}^n]^{n\in N}, T_2' = T_2, \tau_1' = \tau_1, \tau_2' = \tau_2 \cdot l\langle S\rangle$. By Lemma B.8 we have the following subcases:
      (i) If $\mathcal{A}[\,]^{n\in N} - \tau_1 = \mathcal{A}'[\,]^{n\in N'}$, then we have $T_1 - \tau_1 = \mathcal{A}'[\bigoplus_{i\in I_n} !l_i^n\langle S_i^n\rangle.T_i^n]^{n\in N'}$ and $T_1' - \tau_1' = T_1' - \tau_1 = \mathcal{A}'[T_{k_n}^n]^{n\in N'}$. By $T_1 - \tau_1 \bowtie T_2 - \tau_2$, we get

$$T_2 - \tau_2 \;=\; \overline{\mathcal{A}'[\bigoplus_{i\in I_n} !l_i^n\langle S_i^n\rangle.T_i^n]^{n\in N'}} \;=\; \overline{\mathcal{A}'}[\&_{i\in I_n}?l_i^n(S_i^n).\overline{T_i}^n]^{n\in N'}$$

      which implies

$$T_2' - \tau_2' \;=\; T_2 - \tau_2 \cdot l\langle S\rangle \;=\; \overline{\mathcal{A}'}[\&_{i\in I_n}?l_i^n(S_i^n).\overline{T_i}^n]^{n\in N'} - l\langle S\rangle \;=\; \overline{\mathcal{A}'}[\overline{T_{k_n}}^n]^{n\in N'}.$$

      We conclude $T_1' - \tau_1' \bowtie T_2' - \tau_2'$.
      (ii) If $\mathcal{A}[\,]^{n\in N} - \tau_1 = [\,]^{n_0} - \tau_0$, then we have $T_1 - \tau_1 = \bigoplus_{i\in I_{n_0}} !l_i^{n_0}\langle S_i^{n_0}\rangle.T_i^{n_0} - \tau_0$ and $T_1' - \tau_1' = T_{k_{n_0}}^{n_0} - \tau_0$. By $T_1 - \tau_1 \bowtie T_2 - \tau_2$, we get

$$T_2 - \tau_2 \;=\; \overline{\bigoplus_{i\in I_{n_0}} !l_i^{n_0}\langle S_i^{n_0}\rangle.T_i^{n_0} - \tau_0} \;=\; \&_{i\in I_{n_0}}?l_i^{n_0}(S_i^{n_0}).\overline{T_i^{n_0} - \tau_0}$$

      which implies

$$T_2' - \tau_2' \;=\; \&_{i\in I_{n_0}}?l_i^{n_0}(S_i^{n_0}).\overline{T_i^{n_0} - \tau_0} - l\langle S\rangle \;=\; \overline{T_{k_{n_0}}^{n_0} - \tau_0}.$$

      We conclude $T_1' - \tau_1' \bowtie T_2' - \tau_2'$.

(b) If the applied rule is [TR-IN]: $T_2 = \&_{i\in I}?l_i(S_i).T_i$ and $\tau_2 = l_k\langle S\rangle \cdot \tau$ and $S_k \leqslant_{\mathsf{a}} S$, and $k \in I$. As in the proof of case (1b), we can shown that $T_2 - \tau_2 = T_2' - \tau_2'$. Since $T_1' - \tau_1' = T_1 - \tau_1$ we conclude $T_1' - \tau_1' \bowtie T_2' - \tau_2'$. □

Theorem 4.13. (Subject reduction for asynchronous processes) If $\Gamma \vdash_{\mathsf{a}} P \triangleright \Delta$ and Δ is balanced and $P \to_{\mathsf{a}}^* Q$, then there is Δ' such that $\Delta \Rightarrow^* \Delta'$ and $\Gamma \vdash_{\mathsf{a}} Q \triangleright \Delta'$.

Proof. We first prove that if $\Gamma \vdash_{\mathsf{a}} P \triangleright \Delta$ and Δ is balanced and $P \to_{\mathsf{a}} Q$, then there is Δ' such that $\Delta \Rightarrow \Delta'$ and $\Gamma \vdash_{\mathsf{a}} Q \triangleright \Delta$. The proof is by induction on the derivation of $P \to_{\mathsf{a}} Q$.

We only consider some interesting rules of Table 11.

(1) Case [R-SEND-ASYNC]:

$$\Gamma \vdash_{\mathsf{a}} ab \blacktriangleright h \mid a!l\langle c\rangle.P \triangleright \Delta \tag{B.4}$$

By applying Lemma B.4.5 to (B.4), we get

$$\Gamma \vdash_{\mathsf{a}} ab \blacktriangleright h \triangleright \Delta_1 \tag{B.5}$$

$$\Gamma \vdash_{\mathsf{a}} a!l\langle c\rangle.P \triangleright \Delta_2 \tag{B.6}$$

where $\Delta = \Delta_1, \Delta_2$. By applying Lemma B.4.4 to (B.6), we have

$$\Delta_2 = \Delta_2', a : T, c : S \qquad !l\langle S\rangle.T' \leqslant_{\mathsf{a}} T$$
$$\Gamma \vdash_{\mathsf{a}} P \triangleright \Delta_2', a : T' \tag{B.7}$$

By applying Lemma B.4.9 and B.4.10 and rule [T-MESSAGE-Q] to (B.5), we get

$$\Delta_1 = \Delta_1', ab : \tau$$
$$\Gamma \vdash_{\mathsf{a}} ab \blacktriangleright h \cdot l\langle c\rangle \triangleright \Delta_1', ab : \tau \cdot l\langle S\rangle, c : S \tag{B.8}$$

By rule [SUB-SEL] if $\mathcal{A}$ is just one hole and by rule [SUB-PERM-ASYNC] if $\& \in \mathcal{A}$,

$$T = \mathcal{A}[\textstyle\bigoplus_{i\in I_n} !l_i^n\langle S_i^n\rangle.T_i^n]^{n\in N}$$

and for all $n \in N$ there is $i_n \in I_n$ such that $l_{i_n}^n = l$, $S_{i_n}^n \leqslant_{\mathsf{a}} S$, and $T' \leqslant_{\mathsf{a}} \mathcal{A}[T_{i_n}^n]^{n\in N}$. By applying [T-SUB] to (B.7), we derive

$$\Gamma \vdash_{\mathsf{a}} P \triangleright \Delta_2', a : \mathcal{A}[T_{i_n}^n]^{n\in N} \tag{B.9}$$

By applying [T-PAR] to (B.8) and (B.9), we derive

$$\Gamma \vdash_{\mathsf{a}} ab \blacktriangleright h \cdot l\langle c\rangle \mid P \triangleright \Delta'$$

where $\Delta' = \Delta_1', \Delta_2', ab : \tau \cdot l\langle S\rangle, c : S, a : \mathcal{A}[T_{i_n}^n]^{n\in N}$. By [TR-OUT], we have

$$a : T, ab : \tau \Rightarrow a : \mathcal{A}[T_{i_n}^n]^{n\in N}, ab : \tau \cdot l\langle S\rangle$$

which implies $\Delta \Rightarrow \Delta_1', \Delta_2', ab : \tau \cdot l\langle S\rangle, c : S, a : \mathcal{A}[T_{i_n}^n]^{n\in N}$.

(2) Case [R-RECEIVE-ASYNC]:

$$\Gamma \vdash_{\mathsf{a}} ab \blacktriangleright l_k\langle c\rangle \cdot h \mid \sum_{i\in I} b?l_i(x_i).P_i \triangleright \Delta \tag{B.10}$$

By applying Lemma B.4.5 to (B.10), we derive

$$\Gamma \vdash_{\mathsf{a}} ab \blacktriangleright l_k\langle c\rangle \cdot h \triangleright \Delta_1 \tag{B.11}$$

$$\Gamma \vdash_{\mathsf{a}} \sum_{i\in I} b?l_i(x_i).P_i \triangleright \Delta_2 \tag{B.12}$$

where $\Delta = \Delta_1, \Delta_2$. By applying Lemma B.6 to (B.11), we have

$$\Delta_1 = \Delta_1', ab : l_k\langle S'\rangle \cdot \tau, c : S$$
$$S' \leqslant_{\mathsf{a}} S \tag{B.13}$$
$$\Gamma \vdash_{\mathsf{a}} ab \blacktriangleright h \triangleright \Delta_1', ab : \tau \tag{B.14}$$

By applying Lemma B.4.3 to (B.12), we have

$$\Delta_2 = \Delta_2', b : T \qquad T = \&_{j\in J}?l_j(S_j).T_j \qquad J \subseteq I$$
$$\forall j \in J : \Gamma \vdash_{\mathsf{a}} P_j \triangleright \Delta_2', b : T_j, x_j : S_j \tag{B.15}$$

Since Δ is balanced, $T - l_k\langle S'\rangle \cdot \tau$ is defined, and this implies $k \in J$ and

$$S_k \leqslant_{\mathsf{a}} S' \tag{B.16}$$

By applying Lemma B.5 to (B.15), we have

$$\Gamma \vdash_{\mathsf{a}} P_k\{^c/_{x_k}\} \triangleright \Delta_2', b : T_k, c : S_k \tag{B.17}$$

By (B.13) and (B.16), we have $S_k \leqslant_{\mathsf{a}} S' \leqslant_{\mathsf{a}} S$. Hence by applying rule [T-SUB] to (B.17), we derive

$$\Gamma \vdash_{\mathsf{a}} P_k\{^c/_{x_k}\} \triangleright \Delta_2', b : T_k, c : S \tag{B.18}$$

By applying rule [T-PAR] to (B.14) and (B.18) we derive

$$\Gamma \vdash_{\mathsf{a}} ab \blacktriangleright h \mid P_k\{^c/_{x_k}\} \triangleright \Delta_1', \Delta_2', ab : \tau, b : T_k, c : S$$

By rule [TR-IN], we derive $ab : l_k\langle S'\rangle \cdot \tau, b : T \Rightarrow ab : \tau, b : T_k$ which implies

$$\Delta \Rightarrow \Delta_1', \Delta_2', ab : \tau, b : T_k, c : S$$

(3) Case [R-CONTEXT]: We only illustrate the case when the context is a channel restriction. Let $\Gamma \vdash_{\mathsf{a}} (\nu ab)P \triangleright \Delta$, then

$$\Gamma \vdash_{\mathsf{a}} P \triangleright \Delta, a : T_1, b : T_2, ba : \tau_1, ab : \tau_2$$

and $T_1 - \tau_1 \bowtie T_2 - \tau_2$ by Lemma B.4.8. If $P \to_{\mathsf{a}}^* P'$, then by induction there is Δ' such that $\Gamma \vdash_{\mathsf{a}} P \triangleright \Delta'$ and

$$\Delta, a : T_1, b : T_2, ba : \tau_1, ab : \tau_2 \Rightarrow^* \Delta'$$

where Δ' is balanced. This implies $\Delta' = \Delta'', a : T_1', b : T_2', ba : \tau_1', ab : \tau_2'$ and $T_1' - \tau_1' \bowtie T_2' - \tau_2'$ by Lemma 4.12. By rule [T-NEW-ASYNC] we derive

$$\Gamma \vdash_{\mathsf{a}} (\nu ab)P \triangleright \Delta''$$

We have proved that if $\Gamma \vdash_{\mathsf{a}} P \triangleright \Delta$ and Δ is balanced and $P \to_{\mathsf{a}} Q$, then there is Δ' such that $\Delta \Rightarrow \Delta'$ and $\Gamma \vdash_{\mathsf{a}} Q \triangleright \Delta$. Now, the main statement of the theorem can be easily proved by induction on the length of the sequence of transitions $P \to_{\mathsf{a}}^* Q$. □

As in the case of synchronous subtyping we can show:

Lemma B.9. *If $\Gamma, X : \langle T\rangle \vdash_{\mathsf{a}} C[X\langle a\rangle] \triangleright \Delta$, $\Gamma \vdash_{\mathsf{a}} P \triangleright \{a : T\}$, and X does not occur free in C, then $\Gamma \vdash_{\mathsf{a}} C[P] \triangleright \Delta$.*

Proof. The proof is similar to the proof of Lemma A.5. The only case in which the proof differs is $C = (\nu cd)C'$. Let

$$\Gamma, X : \langle T \rangle \vdash_{\mathsf{a}} (\nu cd)C'[X\langle a \rangle] \triangleright \Delta \tag{B.19}$$

$$\Gamma \vdash_{\mathsf{a}} P \triangleright \{a : T\} \tag{B.20}$$

By applying Lemma B.4.8 to (B.19), we have

$$\Gamma, X : \langle T \rangle \vdash_{\mathsf{a}} C'[X\langle a \rangle] \triangleright \Delta, c : T, d : T', cd : \tau, dc : \tau' \text{ and } T - \tau \bowtie T' - \tau' \tag{B.21}$$

By induction, (B.20) and (B.21) together imply $\Gamma \vdash_{\mathsf{a}} C'[P] \triangleright \Delta, c : T, d : T', cd : \tau, dc : \tau'$ and $T - \tau \bowtie T' - \tau'$. By applying [T-NEW-ASYNC], we derive

$$\Gamma \vdash_{\mathsf{a}} (\nu cd)C'[P] \triangleright \Delta$$

□

Theorem 4.15. The asynchronous subtyping relation is sound for the asynchronous calculus.

Proof. The proof is similar to that of Theorem 2.4, using Lemma B.9 and Corollary 4.14. □

Appendix C. Proofs of Section 5

Proposition C.1. $\& \notin \mu\mathbf{t}.T$ if and only if $\& \notin T\{^{\mu\mathbf{t}.T}/_{\mathbf{t}}\}$.

Proof. ($\Longrightarrow$) Assume $\& \notin \mu\mathbf{t}.T$, for some derivation $\mathcal{D}$. We observe that $\mathcal{D}$ can only conclude by the rule for recursion on page 28, and from its premise we have a derivation $\mathcal{D}'$ proving $\& \notin T$. We can inductively rewrite $\mathcal{D}'$ into a derivation proving $\& \notin T\{^{\mu\mathbf{t}.T}/_{\mathbf{t}}\}$, by replacing *(i)* each occurrence of $\mathbf{t}$ with $\mu\mathbf{t}.T$, and *(ii)* each instance of the axiom $\& \notin \mathbf{t}$ (which, after the replacement *(i)*, has become $\& \notin \mu\mathbf{t}.T$) with $\mathcal{D}$ above. We conclude $\& \notin T\{^{\mu\mathbf{t}.T}/_{\mathbf{t}}\}$.

($\Longleftarrow$). The proof for this case is similar to the proof of Proposition B.1, in the $\Longleftarrow$ direction, using $\& \notin T$ and its rules defined on page 28. [3] □

Proposition C.2. $\& \in T$ holds if and only if $\& \notin T$ does *not* hold.

Proof. ($\Longrightarrow$). By induction on the derivation of $\& \in T$. The base case, with $T = \&_{i \in I} ?l_i(S_i).T_i$, is immediate: no rule yields $\& \notin T$. In the inductive case with $T = \bigoplus_{i \in I} !l_i\langle S_i \rangle.T_i$, for all $i \in I$ we have a premise $\& \in T_i$, and thus (by the induction hypothesis) $\& \notin T_i$ does *not* hold: hence, $\nexists i \in I$ such that $\& \notin T_i$ holds, and we conclude that $\& \notin T$ does *not* hold by any rule. In the inductive case with $T = \mu\mathbf{t}.T'$, we have the premise $\& \in T'$, and thus (by the induction hypothesis) $\& \notin T'$ does *not* hold: we conclude that $\& \notin T$ does *not* hold by any rule.

($\Longleftarrow$). We prove $\& \in T$ by induction on T, examining the cases where $\& \notin T$ does *not* hold. If $T = \&_{i \in I} ?l_i(S_i).T_i$, we conclude by the axiom on page 22. If $T = \bigoplus_{i \in I} !l_i\langle S_i \rangle.T_i$ and $\forall i \in I : \& \notin T_i$ does *not* hold, by the induction hypothesis we have $\forall i \in I : \& \in T_i$; therefore, we conclude $\& \in T$ (By the rule for selection on page 22). If $T = \mu\mathbf{t}.T'$ and $\& \notin T'$ does *not* hold, by the induction hypothesis we have $\& \in T'$: therefore, we conclude $\& \in T$ (by the rule for recursion on page 22). □

[3] Notably, the case discussed in footnote 2 is now *not* absurd.