Denotational and Operational Preciseness of Subtyping: A Roadmap

Dedicated to Frank de Boer on the Occasion of His 60th Birthday

Mariangiola Dezani-Ciancaglini[1], Silvia Ghilezan[2], Svetlana Jakšić[2], Jovanka Pantović[2], and Nobuko Yoshida[3(✉)]

[1] Università di Torino, Turin, Italy
[2] Univerzitet u Novom Sadu, Novi Sad, Serbia
[3] Imperial College London, London, UK
n.yoshida@imperial.ac.uk

Abstract. The notion of subtyping has gained an important role both in theoretical and applicative domains: in lambda and concurrent calculi as well as in object-oriented programming languages. The soundness and the completeness, together referred to as the preciseness of subtyping, can be considered from two different points of view: denotational and operational. The former preciseness is based on the denotation of a type, which is a mathematical object describing the meaning of the type in accordance with the denotations of other expressions from the language. The latter preciseness has been recently developed with respect to type safety, i.e. the safe replacement of a term of a smaller type when a term of a bigger type is expected.

The present paper shows that standard proofs of operational preciseness imply denotational preciseness and gives an overview on this subject.

1 Introduction

A subtyping relation is a pre-order (reflexive and transitive relation) on types that validates the principle: if σ is a subtype of τ (notation $\sigma \leq \tau$), then a term of type σ may be provided whenever a term of type τ is needed; see Pierce [35] (Chap. 15) and Harper [20] (Chap. 23).

Partly supported by COST IC1201 BETTY and DART bilateral project between Italy and Serbia.

M. Dezani-Ciancaglini—Partly supported by EU H2020-644235 Rephrase project, EU H2020-644298 HyVar project, ICT COST Actions IC1402 ARVI and Ateneo/CSP project RunVar.

S. Ghilezan et al.—Partly supported by ON 174026 and III 44006 projects of the Ministry of Education, Science and Technological Development, Republic of Serbia.

N. Yoshida—Partly supported by EPSRC EP/K011715/1, EP/K034413/1, and EP/L00058X/1, and EU Project FP7-612985 UpScale.

E. Ábrahám et al. (Eds.): de Boer Festschrift, LNCS 9660, pp. 155–172, 2016.
DOI: 10.1007/978-3-319-30734-3_12

In this paper we will discuss key properties of subtyping, i.e. denotational and operational preciseness. We will introduce these notions in the next two paragraphs.

Denotational Preciseness. A usual approach to preciseness of subtyping for a calculus is to consider the interpretation of a type σ (notation $[\![\sigma]\!]$) to be a set that describes the meaning of the type in accordance with the denotations of the terms of the calculus, in general a subset of the domain of a model of the calculus.

> A subtyping relation is denotationally sound when $\sigma \leq \tau$ implies $[\![\sigma]\!] \subseteq [\![\tau]\!]$ and denotationally complete when $[\![\sigma]\!] \subseteq [\![\tau]\!]$ implies $\sigma \leq \tau$.
> A subtyping relation is denotationally precise if it is both denotationally sound and denotationally complete.

This well-established powerful technique is applied to the pure λ-calculus with arrow and intersection types by Barendregt et al. [4], to a call-by-value λ-calculus with arrow, intersection and union types by van Bakel et al. [2] and by Ishihara and Kurata [25], to a wide class of calculi with arrow, union and pair types by Vouillon [38], and to a concurrent λ-calculus by Dezani and Ghilezan [15]. More recently denotational preciseness was studied for binary sessions [11] and synchronous multiparty sessions [16].

Operational Preciseness. Operational soundness is just the key principle mentioned at the beginning of this section: if $\sigma \leq \tau$, then a term of type σ may be provided whenever a term of type τ is needed. As a simple example $\mathsf{nat} \leq \mathsf{real}$ and a natural number can always play the role of a real number. Operational completeness requires that, if $\sigma \not\leq \tau$, then there are

- a context expecting a term of type τ and
- a term of type σ

such that this context filled with this term behaves badly. As a simple example $\mathsf{nat} \not\leq \mathsf{bool}$, the negation $\neg$ requires a boolean argument and the term $\neg 5$ is stuck.

To define formally operational soundness and completeness we need a boolean predicate bad on terms, standard typing judgements $\Gamma \vdash M : \sigma$ (where Γ is a mapping from variables to types and M is a term) and evaluation contexts C.

> A subtyping relation is operationally sound when $\sigma \leq \tau$ implies that if (for some ρ) $x : \tau \vdash C[x] : \rho$ and $\vdash M : \sigma$, then $\mathsf{bad}(C[M])$ is false, for all C and M.
> A subtyping relation is operationally complete when $\sigma \not\leq \tau$ implies that $x : \tau \vdash C[x] : \rho$ and $\vdash M : \sigma$ and $\mathsf{bad}(C[M])$, for some ρ, C and M.
> A subtyping relation is operationally precise if it is both operationally sound and operationally complete.

Operational soundness immediately follows from the subject reduction theorem, when the subtyping is used in a subsumption rule. A general methodology to prove operational completeness is the following one:

- **[Step 1]** Characterise the negation of the subtyping relation by inductive rules.
- **[Step 2]** For each type σ define a *characteristic context* C_σ, which behaves well when filled with terms of type σ.
- **[Step 3]** For each type σ define a *characteristic term* M_σ, which has only the types greater than or equal to σ.
- **[Step 4]** Show that if $\sigma \not\leq \tau$, then $\mathsf{bad}(C_\tau[M_\sigma])$.

These four steps are the guideline of the proofs in the literature, as we will illustrate in this paper.

Background and Related Work. Ligatti et al. [27] first define operational preciseness of subtyping and apply it to subtyping iso-recursive types. They consider a typed λ-calculus enriched with naturals, reals, pair and case constructors/destructors, and roll/unroll. The predicate $\mathsf{bad}(M)$ holds when M reduces to a stuck term, i.e. to an irreducible term which is not a value. They propose new algorithmic rules for subtyping iso-recursive types and show that they are operationally precise.

Dezani and Ghilezan [15] adapt the ideas of Ligatti et al. [27] to the setting of the concurrent λ-calculus with intersection and union types of [14]. For the operational preciseness they take the view that evaluation of well-typed terms always terminates. This means that the predicate bad coincides with non termination. In this calculus applicative contexts are enough. Notably, soundness and completeness are made more operational by asking that some applications converge instead of being typable. To sum up, the definition of operational preciseness becomes:

> A subtyping $\leq$ is operationally precise when $\sigma \leq \tau$ if and only if there are no closed terms M, N such that ML converges for all closed terms L of type τ and N has type σ and MN diverges.

The main result of this paper is the operational preciseness of the subtyping induced by the standard set theoretic interpretation of arrow, intersection and union types.

Chen et al. [11] first give a general formulation of preciseness for session calculi, where processes are typed by sets of pairs (channels, session types) [22]. The session types prescribe how the channels can be used for communications. The calculus of processes includes an error process and $\mathsf{bad}(P)$ holds when process P reduces to error. The typing judgements for closed processes are of the form $\vdash P \triangleright \{a : T\}$, assuring that the process P has a single free channel a whose type is T. The judgement $\vdash C[a : T] \triangleright \emptyset$ means that filling the hole of C with any process P typed by $a : T$ produces a well-typed closed process. We get:
A subtyping $\leq$ is precise when, for all session types T and S:

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

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. The first result of [11] is that the well-known session subtyping, the branching-selection subtyping [13], is sound and complete for the synchronous dyadic calculus. Next, the authors 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 $T \leq S$ is not derivable by the subtyping. They propose an asynchronous subtyping system (inspired by [32]) which is sound and complete for the asynchronous dyadic calculus. The method gives a general guidance to design rigorous channel-based subtypings respecting desired safety properties.

Dezani et al. [16] consider the synchronous version [26] of the multiparty session calculus in [12,23]. For the operational preciseness they take the view that well-typed sessions never get stuck. Therefore the predicate bad is true for processes which cannot reduce, but contain pending communications. The preciseness of the branching-selection subtyping [13] is shown using a novel notion of characteristic global type.

A framework which is closely related to the above described works is semantic subtyping. In semantic subtyping, each type is interpreted as the set of values having that type and subtyping is subset inclusion between type interpretations [10]. This gives a precise subtyping as soon as the calculus allows to distinguish operationally values of different types.

Semantic subtyping was first proposed by Castagna and Benzaken through the development of the $\mathbb{C}$Duce project [17]. $\mathbb{C}$Duce is a modern XML-oriented functional language. Distinctive features of $\mathbb{C}$Duce are a powerful pattern matching, first class functions, over-loaded functions, a very rich type system (with arrow, sequence, pair, record, intersection, union, difference type constructs), precise type inference for patterns and error localisation, and a natural interpretation of types as sets of values. It is enriched also with some important implementation aspects: in particular, a dispatch algorithm that demonstrates how static type information can be used to obtain very efficient compilation schemas.

Semantic subtyping has been also studied in [8] for a π-calculus with a patterned input and in [9] 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 [8] and session types in [9]. Semantic subtyping is precise for the calculi of [8,9,17], thanks to the type case constructor in [17], and to the blocking of inputs for values of "wrong" types in [8,9].

Bonsangue et al. [6] recently have developed an elegant coalgebraic foundation for coinductive types, which gives a sound and complete characterisation of semantic subtyping in terms of inclusion of maximal traces.

Outline. Sections 2 and 3 deal with typed extensions of λ-calculus, and discuss preciseness of iso-recursive and intersection/union types, respectively. Session calculi are considered in Sects. 4 and 5. Section 4 is devoted to synchronous

and asynchronous binary types, Sect. 5 instead to synchronous multiparty types. Section 6 shows how the existence of characteristic terms as defined in **[Step 3]** implies denotational preciseness. Section 7 concludes with some directions for further work.

2 Iso-Recursive Types

In [27] the authors consider a typed λ-calculus enriched with naturals, reals, pair and case constructors/destructors, and roll/unroll. The syntax of types, terms and values of this calculus (dubbed $L^{\to\mu}_{+\times}$) is given in Fig. 1.

$$\sigma ::= \mathsf{nat} \mid \mathsf{real} \mid \sigma \to \tau \mid \sigma \times \tau \mid \sigma + \tau \mid \mu \mathbf{t}.\sigma \mid \mathbf{t}$$

$$M ::= \mathsf{n} \mid \mathsf{r} \mid \mathsf{succ}(M) \mid \mathsf{neg}(M) \mid \mathsf{fun}\ f(x:\sigma):\tau\ =\ M \mid MM \mid x \mid (M,M) \mid M.\mathsf{fst} \mid M.\mathsf{snd} \mid \mathsf{inl}_\sigma(M) \mid \mathsf{inr}_\sigma(M) \mid \mathsf{case}\ M\ \mathsf{of}\ \mathsf{inl}\ x \Rightarrow M_1\ \mathsf{else}\ \mathsf{inr}\ y \Rightarrow M_2 \mid \mathsf{roll}(M) \mid \mathsf{unroll}(M)$$

$$V ::= \mathsf{n} \mid \mathsf{r} \mid \mathsf{fun}\ f(x:\sigma):\tau\ =\ M \mid (V,V) \mid \mathsf{inl}_\sigma(V) \mid \mathsf{inr}_\sigma(V) \mid \mathsf{roll}(V)$$

Fig. 1. Types, terms and values of $L^{\to\mu}_{+\times}$.

The operational semantics of $L^{\to\mu}_{+\times}$ is call-by-value. The operator succ reduces only when the argument is a natural and unroll is the left inverse of roll. The remaining reduction rules are standard.

The most interesting subtyping rule tells that $\mu\mathbf{t}.\sigma$ is a subtype of $\mu\mathbf{t}.\tau$ if we can derive from $\mu\mathbf{t}.\sigma \le \mu\mathbf{t}.\tau$ that their unfolded versions are in the subtype relation. More precisely:

$$\frac{\Sigma, \mu\mathbf{t}.\sigma \le \mu\mathbf{t}.\tau \vdash \sigma[\mu\mathbf{t}.\sigma/\mathbf{t}] \le \tau[\mu\mathbf{t}.\tau/\mathbf{t}]}{\Sigma \vdash \mu\mathbf{t}.\sigma \le \mu\mathbf{t}.\tau}$$

where Σ is a set of subtyping judgments. The type system is as expected, in particular roll and unroll correspond to fold and unfold of recursive types.

The core of the completeness proof is the construction of characteristic contexts and terms for closed types, as discussed in the Introduction. This construction is delicate since some types (for example $\mu\mathbf{t}.\mathbf{t}$) are not inhabited. The type inhabitation is decidable and every non inhabited type is subtype of all types. Figure 2 shows some of the characteristic contexts and terms for the types of [27]. Notice that in that paper they are used in the proof without grouping them in a unique definition. We omit the case of the sum type being similar to that of the product type. Also, the characteristic contexts and terms for recursive types are missing, since they are quite tricky depending on the external constructor obtained by unfolding the types.

For example $\mathsf{nat} \to \mathsf{nat} \not\le \mathsf{real} \to \mathsf{nat}$. The characteristic context of $\mathsf{real} \to \mathsf{nat}$ is $C_{\mathsf{nat}}[[\ ]M_{\mathsf{real}}] = \mathsf{succ}([\ ]2.5)$. The characteristic term of $\mathsf{nat} \to \mathsf{nat}$ is

$$\mathsf{fun}\ f(x:\mathsf{nat}):\mathsf{nat}\ =\ (\mathsf{fun}\ g(y:\mathsf{nat}):\mathsf{nat}\ =\ M_{\mathsf{nat}})(C_{\mathsf{nat}}[x]),$$

type σ	characteristic context C_σ	characteristic term M_σ
nat	succ[ ]	5
real	neg[ ]	2.5
$\tau_1 \to \tau_2$	$C_{\tau_2}[[\,]M_{\tau_1}]$	fun $f(x:\tau_1):\tau_2 \;=\; M$
$\tau_1 \times \tau_2$	$(C_{\tau_1}[[\,]\text{.fst}], C_{\tau_2}[[\,]\text{.snd}])$	(M_{τ_1}, M_{τ_2})

Fig. 2. Characteristic contexts and terms, where $M = (\mathsf{fun}\ g(y:\tau):\tau_2 \;=\; M_{\tau_2})(C_{\tau_1}[x])$ and τ is the type of $C_{\tau_1}[x]$ when x has type τ_1.

i.e.

$$\mathsf{fun}\ f(x:\mathsf{nat}):\mathsf{nat} \;=\; (\mathsf{fun}\ g(y:\mathsf{nat}):\mathsf{nat} \;=\; 5)(\mathsf{succ}\ x).$$

The term $C_{\mathsf{real}\to\mathsf{nat}}[M_{\mathsf{nat}\to\mathsf{nat}}]$ is then

$$\mathsf{succ}((\mathsf{fun}\ f(x:\mathsf{nat}):\mathsf{nat} \;=\; (\mathsf{fun}\ g(y:\mathsf{nat}):\mathsf{nat} \;=\; 5)(\mathsf{succ}\ x))2.5).$$

This term reduces to

$$\mathsf{succ}((\mathsf{fun}\ g(y:\mathsf{nat}):\mathsf{nat} \;=\; 5)(\mathsf{succ}\ 2.5))$$

which is stuck, since succ 2.5 is stuck.

The main result of [27] is:

Theorem 1. *The subtyping of $L^{\to\mu}_{+\times}$ is operationally precise.*

3 Intersection and Union Types

In this section, we present and discuss the results from [15] on denotational and operational preciseness of the subtyping relation in the setting of the concurrent λ-calculus with intersection and union types (dubbed $\lambda_{\oplus\|}$) introduced in [14]. The syntax of types, terms, values, and total values of this calculus is given in Fig. 3. The only atomic type is the universal type ω. There are both call-by-name variables (ranged over by x) and call-by-value variables (ranged over by v). The constructor $\oplus$ is the non-deterministic choice and the constructor $\|$ is the parallel operator.

$$\begin{gathered}
\sigma ::= \omega \mid \sigma \to \sigma \mid \sigma \wedge \sigma \mid \sigma \vee \sigma \\
M ::= x \mid v \mid (\lambda x.M) \mid (\lambda v.M) \mid (MM) \mid (M \oplus M) \mid (M \| M) \\
V ::= v \mid \lambda x.M \mid \lambda v.M \mid V \| M \mid M \| V \\
W ::= v \mid \lambda x.M \mid \lambda v.M \mid W \| W
\end{gathered}$$

Fig. 3. Types, terms, values, and total values of $\lambda_{\oplus\|}$.

The reduction relation formalises the behaviour of a machine which evaluates in a synchronous way parallel compositions, until a value is produced. Partial values, i.e. values which are not total, can be further evaluated, and this is essential for applications of a call-by-value abstraction (rule $(\beta_v\|)$). The reduction rules which enable this behaviour are the following

$$(\mu_v)\ \frac{N \longrightarrow N' \quad N \notin \mathsf{Val}}{(\lambda v.M)N \longrightarrow (\lambda v.M)N'} \qquad (\beta_v\|)\ \frac{V \longrightarrow V' \quad V \in \mathsf{Val}}{(\lambda v.M)V \longrightarrow M[V/v]\|(\lambda v.M)V'}$$

According to [14] a term is *convergent* if all reduction paths reach values.

The type system with intersection and union types is dually reflecting the conjunctive and disjunctive operational semantics of $\|$ and $\oplus$. The subtyping relation on $Type$, the set of all types, is the smallest pre-order such that

1. $\langle Type, \leq \rangle$ is a distributive lattice, where $\wedge$ is the meet, $\vee$ is the join, ω is the top;
2. the arrow satisfies
   (a) $\sigma \to \omega \leq \omega \to \omega$;
   (b) $(\sigma \to \rho) \wedge (\sigma \to \tau) \leq \sigma \to \rho \wedge \tau$;
   (c) $\sigma \geq \sigma', \tau \leq \tau' \Rightarrow \sigma \to \tau \leq \sigma' \to \tau'$.

Notice that the standard axiom $(\sigma \to \rho) \wedge (\tau \to \rho) \leq \sigma \vee \tau \to \rho$ [2,25] is unsound for $\lambda_{\oplus\|}$, as proven in [14].

Regarding operational preciseness, divergent terms are the ones that are not convergent and the predicate bad coincides with divergence. Closed convergent and divergent terms are completely characterised by the types $\omega \to \omega$ and ω, respectively [14].

As said in the Introduction, it is enough to consider applicative context, that we call *test tems*. Figure 4 gives test and characteristic terms, where $\mathbf{I} = \lambda x.x$ and $\Omega = (\lambda x.xx)(\lambda x.xx)$. For example $M_{\omega\to\omega} = \lambda x.\Omega$ and $N_{\omega\to\omega} = \lambda v.\mathbf{I}$. More interestingly $M_{(\omega\to\omega)\to\omega\to\omega} = \lambda x.((\lambda v.\mathbf{I})x)(\lambda y.\Omega)$ applied to a term returns $\lambda y.\Omega$ only if the term reduces to a value. Similarly $N_{(\omega\to\omega)\to\omega\to\omega} = \lambda v.(\lambda v'.\mathbf{I})(v(\lambda x.\Omega))$ applied to a term which reduces to a value, first applies this term to $\lambda x.\Omega$, and then reduces to $\mathbf{I}$ only if the result of this application reduces to a value too.

The key property of test terms is:

if M is a closed term, then $N_\sigma M$ converges if and only if M has type σ.

As a consequence $\sigma \not\leq \tau$ implies the divergence of $N_\tau M_\sigma$, i.e. $\mathsf{bad}(N_\tau M_\sigma)$.

The denotational preciseness of this subtyping is obtained for the standard set-theoretic interpretation of arrow, intersection and union types. The key tool is the existence of characteristic terms, as shown in Sect. 6.

To sum up, the main result in [15] is:

Theorem 2 (Denotational and Operational Preciseness).

1. *The subtyping of the $\lambda_{\oplus\|}$-calculus is operationally precise.*
2. *The subtyping of the $\lambda_{\oplus\|}$-calculus is denotationally precise.*

type σ	test term N_σ	characteristic term M_σ
ω	$\lambda x.\mathbf{I}$	Ω
$\tau_1 \to \tau_2$	$\lambda v.N_{\tau_2}(vM_{\tau_1})$	$\lambda x.(N_{\tau_1}x)M_{\tau_2}$
$\tau_1 \wedge \tau_2$	$\lambda x.(N_{\tau_1}x \oplus N_{\tau_2}x)$	$M_{\tau_1} \| M_{\tau_2}$
$\tau_1 \vee \tau_2$	$\lambda v.(N_{\tau_1}v \| N_{\tau_2}v)$ where $\tau_1 \vee \tau_2 \neq \omega$	$M_{\tau_1} \oplus M_{\tau_2}$

Fig. 4. Test and characteristic terms.

4 Binary Session Types

This section presents results from [11] stating that the well-known branching-selection subtyping (defined in Fig. 7) is precise for the synchronous session calculus. As it happens that this subtyping is incomplete for type-safety for the asynchronous session calculus, the authors propose an asynchronous subtyping relation and prove that it is precise for the asynchronous session calculus.

4.1 Synchronous Session Calculus

A *binary 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. The syntax of the synchronous session calculus is given in Fig. 5. The input process $\sum_{i\in I} u?l_i(x_i).P_i$ waits on channel u for a label l_i and a channel to replace x_i inside P_i ($i \in I$). The output process sends on channel u the label l and the channel u'. The process $\mathtt{def}\ D\ \mathtt{in}\ P$ is a recursive agent and $X\langle\tilde{u}\rangle$ is a recursive variable. 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.

$$P ::= \mathbf{0} \mid X\langle\tilde{u}\rangle \mid \sum_{i\in I} u?l_i(x_i).P_i \mid u!l\langle u'\rangle.P \mid P \oplus P \mid P \mid P \mid \mathtt{def}\ D\ \mathtt{in}\ P \mid (\nu ab)P \mid \mathsf{error}$$
$$u ::= a \mid x \qquad D ::= X(\tilde{x}) = P$$

Fig. 5. Syntax of synchronous processes.

Operational semantics is given by a reduction relation between the synchronous processes. The main rule is

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

sessiontype T	characteristic process $\mathbf{P}(u,T)$
$\mathtt{end}$	0
$\mathbf{t}$	$X_{\mathbf{t}}\langle u\rangle$
$\&_{i\in I}?l_i(S_i).T_i$	$\sum_{i\in I} u?l_i(x).(\mathbf{P}(u,T_i)\,\vert\,\mathbf{P}(x,S_i))$
$\bigoplus_{i\in I}!l_i\langle S_i\rangle.T_i$	$\bigoplus_{i\in I}(\nu ab)(u!l_i\langle a\rangle.\mathbf{P}(u,T_i)\,\vert\,\mathbf{P}(b,\overline{S}_i))$
$\mu\mathbf{t}.S$	$\mathtt{def}\ X_{\mathbf{t}}(x)=\mathbf{P}(x,S)\ \mathtt{in}\ X_{\mathbf{t}}\langle u\rangle$

Fig. 6. Types and characteristic synchronous processes.

$$\text{[SUB-END]}\quad \mathtt{end}\leqslant\mathtt{end} \qquad \text{[SUB-BRA]}\quad \frac{\forall i\in I: S_i\leqslant S_i' \quad T_i\leqslant T_i'}{\&_{i\in I\cup J}?l_i(S_i).T_i\leqslant\&_{i\in I}?l_i(S_i').T_i'} \qquad \text{[SUB-SEL]}\quad \frac{\forall i\in I: S_i'\leqslant S_i \quad T_i\leqslant T_i'}{\bigoplus_{i\in I}!l_i\langle S_i\rangle.T_i\leqslant\bigoplus_{i\in I\cup J}!l_i\langle S_i'\rangle.T_i'}$$

Fig. 7. Synchronous subtyping.

It describes the communication between an output $(a!l_k\langle c\rangle.P)$ and an input $(\sum_{i\in I} b?l_i(x_i).Q_i)$ at two co-channels a and b, where the label l_k is selected and channel c replaces x_k into the k-th input branch (Q_k). Other rules are standard.

The synchronous session calculus includes an error process and $\mathsf{bad}(P)$ holds when process P reduces to error. There are four kinds of processes which generate error: a session with mismatch between corresponding output and input labels, a session where one of two co-channels is missing, a session where two co-channels are both subjects of outputs, and a session where two co-channels are both subjects of inputs.

The syntax of *synchronous session types* is given in Fig. 6. As usual *session duality* [22] plays an important rôle for session types. The function $\overline{T}$, defined below, yields the dual of the session type T.

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

The type system is the standard one for session calculi, see e.g. [13]. The subtyping relation is given in Fig. 7, where the double line in rules indicates that the rules are interpreted *coinductively* [35] (Chap. 21). The type system enjoys the property of subject reduction, which implies operational soundness of the synchronous subtyping.

It can be verified that the relation $\not\trianglelefteq$, presented in Fig. 8, is the negation of the synchronous subtyping.

The characteristic process offering communication T on identifier u for the synchronous calculus, denoted by $\mathsf{P}(u,T)$, is given in Fig. 6.

For type S and channel b, the characteristic context is defined as

$$C_{S,b}=[\,]\,\vert\,\mathbf{P}(b,\overline{S}).$$

$$\frac{T \neq \mathtt{end}}{\mathtt{end} \not\trianglelefteq T}\ [\text{N-END R}] \qquad \frac{T \neq \mathtt{end}}{T \not\trianglelefteq \mathtt{end}}\ [\text{N-END L}] \qquad \&_{i\in I}?l_i(S_i).T_i \not\trianglelefteq \oplus_{j\in J}!l'_j\langle S'_j\rangle.T'_j\ [\text{N-BRASEL}] \qquad \oplus_{j\in J}!l'_j\langle S'_j\rangle.T'_j \not\trianglelefteq \&_{i\in I}?l_i(S_i).T_i\ [\text{N-SELBRA-SYNC}]$$

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

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

Fig. 8. Negation of synchronous subtyping.

Finally, it can be proven that $T \not\trianglelefteq S$ implies

$$\mathsf{bad}((\nu ab)C_{S,b}[\mathbf{P}(a,T)]) = \mathsf{bad}((\nu ab)(\mathbf{P}(a,T) \mid \mathbf{P}(b,\overline{S}))).$$

For example (omitting $\mathbf{0}$ and final $\mathtt{end}$) let $T = !l_1(\mathtt{end}).?l_2(\mathtt{end})$ and $S = ?l_2(\mathtt{end}).!l_1(\mathtt{end})$, then $T \not\trianglelefteq S$. By definition

$$\begin{aligned}\mathbf{P}(a,T) &= (\nu c_1 d_1)(a!l_1\langle c_1\rangle.\mathbf{P}(a,?l_2(\mathtt{end})) \mid \mathbf{P}(d_1,\mathtt{end}))\\ &= (\nu c_1 d_1)(a!l_1\langle c_1\rangle.a?l_2(x).(\mathbf{P}(a,\mathtt{end}) \mid \mathbf{P}(x,\mathtt{end})))\\ &= (\nu c_1 d_1)(a!l_1\langle c_1\rangle.a?l_2(x))\end{aligned}$$

We get $\overline{S} = !l_2(\mathtt{end}).?l_1(\mathtt{end})$ and

$$\begin{aligned}\mathbf{P}(b,\overline{S}) &= (\nu c_2 d_2)(b!l_2\langle c_2\rangle.\mathbf{P}(b,?l_1(\mathtt{end})) \mid \mathbf{P}(d_2,\mathtt{end}))\\ &= (\nu c_2 d_2)(b!l_2\langle c_2\rangle.b?l_1(y).(\mathbf{P}(b,\mathtt{end}) \mid \mathbf{P}(y,\mathtt{end})))\\ &= (\nu c_2 d_2)(b!l_2\langle c_2\rangle.b?l_1(y))\end{aligned}$$

Then

$$\begin{aligned}(\nu ab)C_{S,b}[\mathbf{P}(a,T)] &= (\nu ab)(\mathbf{P}(a,T) \mid \mathbf{P}(b,\overline{S}))\\ &= (\nu ab)((\nu c_1 d_1)(a!l_1\langle c_1\rangle.a?l_2(x)) \mid (\nu c_2 d_2)(b!l_2\langle c_2\rangle.b?l_1(y)))\end{aligned}$$

and this last process reduces to error, since the two co-channels are both subjects of outputs.

In [11], the main result for synchronous subtyping is:

Theorem 3 (Preciseness for Synchronous Session Calculus). *The synchronous subtyping relation is operationally precise for the synchronous session calculus.*

4.2 Asynchronous Session Calculus

The asynchronous session calculus is obtained from the rules for the synchronous ones by extending the synchronous calculus of Fig. 5 with queues:

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

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

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

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

Fig. 9. Negation of asynchronous subtyping.

$$P ::= \ldots \mid ab \blacktriangleright h \qquad h ::= \varnothing \mid l\langle a\rangle \mid h \cdot h.$$

A queue $ab \blacktriangleright h$ is used by channel a to enqueue messages in h and by channel b to dequeue messages from h.

Reduction rules for asynchronous processes are obtained from the rules for the synchronous processes by replacing [R-COM-SYNC] with the following two rules:

$$ab \blacktriangleright h \mid a!l\langle c\rangle.P \longrightarrow ab \blacktriangleright h \cdot l\langle c\rangle \mid P\ [\text{R-SEND-ASYNC}]$$

$$\frac{k \in I}{ab \blacktriangleright l_k\langle c\rangle \cdot h \mid \sum_{i\in I} b?l_i(x_i).P_i \longrightarrow ab \blacktriangleright h \mid P_k\{c/x_k\}}\ [\text{R-RECEIVE-ASYNC}]$$

In presence of queues, reduction to error includes deadlocks, that are sessions with inputs waiting to dequeue messages from queues that will stay empty, and orphan messages, that are messages in queues that will never be received.

To define asynchronous subtyping, the notion of asynchronous context is introduced, that is a sequence of branchings containing indexed holes:

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

The asynchronous subtyping relation is obtained by extending synchronous subtyping relation by the rule:

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

Using this rule we get for example $!l_1(\mathtt{end}).?l_2(\mathtt{end}) \leq ?l_2(\mathtt{end}).!l_1(\mathtt{end})$, which does not hold in the synchronous subtyping, as shown in previous subsection.

The negation rules of asynchronous subtyping are the rules of Fig. 8 excluding rule [N-SELBRA-SYNC], extended by the rules of Fig. 9.

The characteristic process offering communication T on identifier u for the asynchronous calculus, denoted by $\mathbf{P}(u, T)$, is defined as in Fig. 6, but for the case of T being $\bigoplus_{i\in I} !l_i\langle S_i\rangle.T_i$, which becomes:

$$\bigoplus_{i\in I} (\nu ab)(u!l_i\langle a\rangle.\mathbf{P}(u, T_i) \mid \mathbf{P}(b, \overline{S}_i) \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing).$$

For type S and channel b, the characteristic context is defined as

$$C_{S,b} = [\,] \mid \mathbf{P}(b, \overline{S}) \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing.$$

For $T \not\leqslant S$, we can prove that there are $T' \leqslant T$ and $S' \geq S$ such that

$$\mathsf{bad}((\nu ab)(C_{S',b}[\mathbf{P}(a, T'])) = \mathsf{bad}((\nu ab)(\mathbf{P}(a, T') \mid \mathbf{P}(b, \overline{S'}) \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing)).$$

Notice that $S' \geq S$ if and only if $\overline{S'} \leqslant \overline{S}$.

For example let $T = !l_1(\mathtt{end}) \oplus !l_2(\mathtt{end})$ and $S = !l_1(\mathtt{end})$, then $T \not\leqslant S$. By definition

$$\begin{aligned}\mathbf{P}(a, T) &= (\nu c_1 d_1)(a!l_1\langle c_1\rangle.\mathbf{P}(a, \mathtt{end}) \mid \mathbf{P}(d_1, \mathtt{end}) \mid d_1 c_1 \blacktriangleright \varnothing \mid c_1 d_1 \blacktriangleright \varnothing) \oplus \\ &\quad (\nu c_2 d_2)(a!l_2\langle c_2\rangle.\mathbf{P}(a, \mathtt{end}) \mid \mathbf{P}(d_2, \mathtt{end}) \mid d_2 c_2 \blacktriangleright \varnothing \mid c_2 d_2 \blacktriangleright \varnothing) \\ &= (\nu c_1 d_1)(a!l_1\langle c_1\rangle \mid d_1 c_1 \blacktriangleright \varnothing \mid c_1 d_1 \blacktriangleright \varnothing) \oplus \\ &\quad (\nu c_2 d_2)(a!l_2\langle c_2\rangle \mid d_2 c_2 \blacktriangleright \varnothing \mid c_2 d_2 \blacktriangleright \varnothing).\end{aligned}$$

We also get $\overline{S} = ?l_1(\mathtt{end})$ and

$$\mathbf{P}(b, \overline{S}) = b?l_1(y).(\mathbf{P}(b, \mathtt{end}) \mid \mathbf{P}(y, \mathtt{end})) = b?l_1(y).$$

Then

$$\begin{aligned}(\nu ab)C_{S,b}[\mathbf{P}(a, T)] = \;& (\nu ab)(\mathbf{P}(a, T) \mid \mathbf{P}(b, \overline{S}) \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing) \\ = \;& (\nu ab)((\nu c_1 d_1)(a!l_1\langle c_1\rangle \mid d_1 c_1 \blacktriangleright \varnothing \mid c_1 d_1 \blacktriangleright \varnothing) \oplus \\ & (\nu c_2 d_2)(a!l_2\langle c_2\rangle \mid d_2 c_2 \blacktriangleright \varnothing \mid c_2 d_2 \blacktriangleright \varnothing) \mid \\ & \quad b?l_1(y) \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing) \\ \longrightarrow \;& (\nu ab)((\nu c_2 d_2)(a!l_2\langle c_2\rangle \mid d_2 c_2 \blacktriangleright \varnothing \mid c_2 d_2 \blacktriangleright \varnothing) \mid \\ & \quad b?l_1(y) \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright \varnothing) \\ \longrightarrow \;& (\nu ab)(\nu c_2 d_2)(d_2 c_2 \blacktriangleright \varnothing \mid c_2 d_2 \blacktriangleright \varnothing \mid \\ & \quad b?l_1(y) \mid ba \blacktriangleright \varnothing \mid ab \blacktriangleright l_2\langle c_2\rangle) \\ \longrightarrow \;& \mathsf{error}\end{aligned}$$

where the reduction to error is due to the mismatch between the input label l_1 and the label l_2 of the message.

In [11], the main result for asynchronous subtyping is:

Theorem 4 (Preciseness for Asynchronous Subtyping). *The asynchronous subtyping relation is operationally precise for the asynchronous session calculus.*

5 Multiparty Session Types

In [16] the authors show operational and denotational preciseness of the subtyping introduced in [13] for a simplification of the synchronous multiparty session calculus in [26]. The calculus is obtained by eliminating both shared channels for session initiations and session channels for communications inside sessions.

$$P ::= \mathbf{0} \mid X \mid \mathsf{p}?\ell(x).P \mid \mathsf{p}!\ell(\mathsf{e}).P \mid P+P \mid \mathsf{if}\ \mathsf{e}\ \mathsf{then}\ P\ \mathsf{else}\ P \mid \mu X.P$$
$$\mathscr{M} ::= \mathsf{p} \triangleleft P \mid \mathscr{M} \mid \mathscr{M}$$

Fig. 10. Processes and multiparty sessions.

$$\mathsf{S} ::= \mathtt{nat} \mid \mathtt{int} \mid \mathtt{bool}$$
$$\mathsf{G} ::= \mathsf{p} \to \mathsf{q} : \{\ell_i(\mathsf{S}_i).\mathsf{G}_i\}_{i\in I} \mid \mu \mathbf{t}.\mathsf{G} \mid \mathbf{t} \mid \mathtt{end}$$
$$\mathsf{T} ::= \bigwedge_{i\in I} \mathsf{p}?\ell_i(\mathsf{S}_i).\mathsf{T}_i \mid \bigvee_{i\in I} \mathsf{q}!\ell_i(\mathsf{S}_i).\mathsf{T}_i \mid \mu \mathbf{t}.\mathsf{T} \mid \mathbf{t} \mid \mathtt{end}$$

Fig. 11. Sorts, global types and multiparty session types.

A multiparty session is a series of interactions between a fixed number of participants, possibly with branching and recursion, and serves as a unit of abstraction for describing communication protocols. The syntax of processes and multiparty sessions is given in Fig. 10. The values are natural numbers n, integers i, and boolean values true and false. The expressions e are variables or values or expressions built from expressions by applying the operators $\mathtt{succ}, \mathtt{neg}, \neg, \oplus$, or the relation $>$. The input process $\mathsf{p}?\ell(x).P$ waits for an expression with label ℓ from participant p and the output process $\mathsf{q}!\ell(\mathsf{e}).Q$ sends the value of expression e with label ℓ to participant q. The external choice $P + Q$ offers to choose either P or Q. The process $\mu X.P$ is a recursive process. An equi-recursive view is taken, not distinguishing between a process $\mu X.P$ and its unfolding $P\{\mu X.P/X\}$. If $\mathsf{p} \triangleleft P$ is well typed (see typing rules in [16]), then participant p does not occur in process P, since we do not allow self-communications.

The computational rules of multiparty sessions are closed with respect to the structural congruence (defined as expected) and reduction contexts (empty and parallel composition). Here we recall only the main rule [R-COMM] which states that participant q sends the value v choosing label ℓ_j to participant p which offers inputs on all labels ℓ_i with $i \in I$.

$$[\text{R-COMM}]\quad \frac{j \in I \qquad \mathsf{e} \downarrow \mathsf{v}}{\mathsf{p} \triangleleft \sum_{i\in I} \mathsf{q}?\ell_i(x).P_i \mid \mathsf{q} \triangleleft \mathsf{p}!\ell_j(\mathsf{e}).Q \longrightarrow \mathsf{p} \triangleleft P_j\{\mathsf{v}/x\} \mid \mathsf{q} \triangleleft Q}$$

The value v of expression e (notation $\mathsf{e} \downarrow \mathsf{v}$) is as expected, see [16]. The successor operation $\mathtt{succ}$ is defined only on natural numbers, the negation $\mathtt{neg}$ is defined on integers (and then also on natural numbers), and $\neg$ is defined only on boolean values. The internal choice $\mathsf{e}_1 \oplus \mathsf{e}_2$ evaluates either to the value of e_1 or to the value of e_2.

In order to define the operational preciseness of subtyping it is crucial to formalise when a multiparty session contains communications that will never be executed.

$$[\text{SUB-END}]\quad \mathtt{end} \leqslant \mathtt{end} \qquad [\text{SUB-IN}]\quad \frac{\forall i \in I:\quad \mathsf{S}'_i \leq: \mathsf{S}_i \quad \mathsf{T}_i \leqslant \mathsf{T}'_i}{\bigwedge_{i\in I\cup J} \mathsf{p}?\ell_i(\mathsf{S}_i).\mathsf{T}_i \leqslant \bigwedge_{i\in I} \mathsf{p}?\ell_i(\mathsf{S}'_i).\mathsf{T}'_i} \qquad [\text{SUB-OUT}]\quad \frac{\forall i \in I:\quad \mathsf{S}_i \leq: \mathsf{S}'_i \quad \mathsf{T}_i \leqslant \mathsf{T}'_i}{\bigvee_{i\in I} \mathsf{p}!\ell_i(\mathsf{S}_i).\mathsf{T}_i \leqslant \bigvee_{i\in I\cup J} \mathsf{p}!\ell_i(\mathsf{S}'_i).\mathsf{T}'_i}$$

Fig. 12. Subtyping of multiparty session types.

$$[\text{NSUB-ENDL}]\quad \frac{\mathsf{T} \neq \mathtt{end}}{\mathsf{T} \not\leqslant \mathtt{end}} \qquad [\text{NSUB-ENDR}]\quad \frac{\mathsf{T} \neq \mathtt{end}}{\mathtt{end} \not\leqslant \mathsf{T}} \qquad [\text{NSUB-DIFF-PART}]\quad \frac{\mathsf{p} \neq \mathsf{q} \qquad \dagger,\ddagger \in \{?,!\}}{\mathsf{p} \dagger \ell_1(\mathsf{S}_1).\mathsf{T}_1 \not\leqslant \mathsf{q} \ddagger \ell_2(\mathsf{S}_2).\mathsf{T}_2}$$

$$[\text{NSUB-OUT-IN}]\quad \mathsf{p}!\ell_1(\mathsf{S}_1).\mathsf{T}_1 \not\leqslant \mathsf{p}?\ell_2(\mathsf{S}_2).\mathsf{T}_2 \qquad [\text{NSUB-IN-OUT}]\quad \mathsf{p}?\ell_1(\mathsf{S}_1).\mathsf{T}_1 \not\leqslant \mathsf{p}!\ell_2(\mathsf{S}_2).\mathsf{T}_2$$

$$[\text{NSUB-IN-IN}]\quad \frac{\ell_1 \neq \ell_2 \text{ or } \mathsf{S}_2 \not\leq: \mathsf{S}_1 \text{ or } \mathsf{T}_1 \not\leqslant \mathsf{T}_2}{\mathsf{p}?\ell_1(\mathsf{S}_1).\mathsf{T}_1 \not\leqslant \mathsf{p}?\ell_2(\mathsf{S}_2).\mathsf{T}_2} \qquad [\text{NSUB-OUT-OUT}]\quad \frac{\ell_1 \neq \ell_2 \text{ or } \mathsf{S}_1 \not\leq: \mathsf{S}_2 \text{ or } \mathsf{T}_1 \not\leqslant \mathsf{T}_2}{\mathsf{p}!\ell_1(\mathsf{S}_1).\mathsf{T}_1 \not\leqslant \mathsf{p}!\ell_2(\mathsf{S}_2).\mathsf{T}_2}$$

$$[\text{NSUB-INTR}]\quad \frac{\mathsf{T} \not\leqslant \mathsf{T}_1 \text{ or } \mathsf{T} \not\leqslant \mathsf{T}_2}{\mathsf{T} \not\leqslant \mathsf{T}_1 \wedge \mathsf{T}_2} \qquad [\text{NSUB-UNIL}]\quad \frac{\mathsf{T}_1 \not\leqslant \mathsf{T} \text{ or } \mathsf{T}_2 \not\leqslant \mathsf{T}}{\mathsf{T}_1 \vee \mathsf{T}_2 \not\leqslant \mathsf{T}} \qquad [\text{NSUB-INTL-UNIR}]\quad \frac{\forall i \in I\ \forall j \in J\ \mathsf{T}_i \not\leqslant \mathsf{T}'_j}{\bigwedge_{i\in I} \mathsf{T}_i \not\leqslant \bigvee_{j\in J} \mathsf{T}'_j}$$

Fig. 13. Negation of subtyping of multiparty session types.

Definition 1. *A multiparty session* $\mathscr{M}$ is stuck *if* $\mathscr{M} \not\equiv \mathsf{p} \triangleleft \mathbf{0}$ *and there is no multiparty session* $\mathscr{M}'$ *such that* $\mathscr{M} \longrightarrow \mathscr{M}'$. *A multiparty session* $\mathscr{M}$ gets stuck, *notation* $\mathtt{stuck}(\mathscr{M})$, *if it reduces to a stuck multiparty session.*

A stuck multiparty session is a bad multiparty session, i.e. $\mathsf{bad}(\mathscr{M}) = \mathtt{stuck}(\mathscr{M})$.

The type system is the simplification of that in [26] due to the new formulation of the calculus. Figure 11 contains syntax of sorts, global types and session types.

Global types describe the whole conversation scenarios of multiparty sessions. Session types correspond to projections of global types on the individual participants.

Subsorting $\leq:$ on sorts is the minimal reflexive and transitive closure of the relation induced by the rule: $\mathtt{nat} \leq: \mathtt{int}$. Subtyping $\leqslant$ on session types takes into account the contra-variance of inputs, the covariance of outputs, and the standard rules for intersection and union. Figure 12 gives the coinductive subtyping rules.

The proof of operational soundness of subtyping follows from the subsumption rule and the safety theorem of the type system.

The proof of operational completeness comes in four steps as stated in Introduction.

The characterisation of the negation of the subtyping is given in Fig. 13.

session type T	characteristic process $\mathscr{P}(\mathsf{T})$
$\mathtt{end}$	$\mathbf{0}$
$\mathbf{t}$	$X_{\mathbf{t}}$
$\mathsf{p}?\ell(\mathtt{nat}).\mathsf{T}'$	$\mathsf{p}?\ell(x).\mathsf{if}\ \mathtt{succ}(x) > 0\ \mathsf{then}\ \mathscr{P}(\mathsf{T}')\ \mathsf{else}\ \mathscr{P}(\mathsf{T}')$
$\mathsf{p}?\ell(\mathtt{int}).\mathsf{T}'$	$\mathsf{p}?\ell(x).\mathsf{if}\ \mathtt{neg}(x) > 0\ \mathsf{then}\ \mathscr{P}(\mathsf{T}')\ \mathsf{else}\ \mathscr{P}(\mathsf{T}')$
$\mathsf{p}?\ell(\mathtt{bool}).\mathsf{T}'$	$\mathsf{p}?\ell(x).\mathsf{if}\ \neg x\ \mathsf{then}\ \mathscr{P}(\mathsf{T}')\ \mathsf{else}\ \mathscr{P}(\mathsf{T}')$
$\mathsf{p}!\ell(\mathtt{nat}).\mathsf{T}$	$\mathsf{p}!\ell(5).\mathscr{P}(\mathsf{T}')$
$\mathsf{p}!\ell(\mathtt{int}).\mathsf{T}'$	$\mathsf{p}!\ell(-5).\mathscr{P}(\mathsf{T}')$
$\mathsf{p}!\ell(\mathtt{bool}).\mathsf{T}'$	$\mathsf{p}!\ell(\mathsf{true}).\mathscr{P}(\mathsf{T}')$
$\mathsf{T}_1 \wedge \mathsf{T}_2$	$\mathscr{P}(\mathsf{T}_1) + \mathscr{P}(\mathsf{T}_2)$
$\mathsf{T}_1 \vee \mathsf{T}_2$	$\mathsf{if}\ \mathsf{true} \oplus \mathsf{false}\ \mathsf{then}\ \mathscr{P}(\mathsf{T}_1)\ \mathsf{else}\ \mathscr{P}(\mathsf{T}_2)$
$\mu\mathbf{t}.\mathsf{T}'$	$\mu X_{\mathbf{t}}.\mathscr{P}(\mathsf{T}')$

Fig. 14. Characteristic processes.

The characteristic process $\mathscr{P}(\mathsf{T})$ of type T is defined in Fig. 14 by using the operators $\mathtt{succ}$, $\mathtt{neg}$, and $\neg$ to check if the received values are of the right sort and exploiting the correspondence between external choices and intersections, conditionals and unions.

The authors define the characteristic global type $\mathscr{G}(\mathsf{T}, \mathsf{p})$ of type T for participant p, that describes the communications between p and all participants which occur in$\mathsf{T}$ (notation $\mathtt{pt}\{\mathsf{T}\}$). Moreover, after each communication involving p and some $\mathsf{q} \in \mathtt{pt}\{\mathsf{T}\}$, participant q starts a cyclic communication involving all participants in $\mathtt{pt}\{\mathsf{T}\}$ both as receivers and senders. The characteristic context for $\mathsf{p} \triangleleft \mathscr{P}(\mathsf{T})$ is built using the characteristic global type of type T for participant p.

We do not give here the definitions of characteristic global types and characteristic contexts, we only show an example. Let $\mathsf{T} = \mathsf{p}_1!\ell_1(\mathtt{nat}).\mathsf{p}_2!\ell_2(\mathtt{nat})$ and $\mathsf{T}' = \mathsf{p}_2!\ell_2(\mathtt{nat}).\mathsf{p}_1!\ell_1(\mathtt{nat})$. Clearly $\mathsf{T} \not\leq \mathsf{T}'$ and $\mathscr{P}(\mathsf{T}) = \mathsf{p}_1!\ell_1(5).\mathsf{p}_2!\ell_2(5)$. The characteristic context for $\mathsf{p} \triangleleft \mathscr{P}(\mathsf{T})$ is $[\,] \mid \mathsf{p}_1 \triangleleft \mathsf{p}_2?\ell_2(x)... \mid \mathsf{p}_2 \triangleleft \mathsf{p}?\ell_2(x)...$ and the process

$$\mathsf{p} \triangleleft \mathsf{p}_1!\ell_1(5).\mathsf{p}_2!\ell_2(5) \mid \mathsf{p}_1 \triangleleft \mathsf{p}_2?\ell_2(x)... \mid \mathsf{p}_2 \triangleleft \mathsf{p}?\ell_2(x)...$$

is stuck, since participant p wants to send a message to participant p_1, who instead is ready to receive a message from participant p_2, who in turn expects a message from participant p.

The main result of [16] is:

Theorem 5. *The synchronous multiparty session subtyping is operationally precise.*

6 Characteristic Terms for Denotational Preciseness

It is standard [11,15,16,21] to interpret a type σ as the set of closed terms typed by σ, i.e.

$$[\![\sigma]\!] = \{M \mid \vdash M : \sigma\}$$

In this case denotational soundness immediately follows from the subsumption rule. Moreover, the existence of characteristic terms as defined in **[Step 3]** at page 3 implies denotational completeness. By definition characteristic terms enjoy the following key property:

$$\vdash M_\sigma : \tau \text{ implies } \sigma \leqslant \tau.$$

We get denotational completeness, since if $\sigma \not\leqslant \tau$, then $M_\sigma \in [\![\sigma]\!]$, but $M_\sigma \notin [\![\tau]\!]$.

Theorem 6 (Denotational Preciseness). *The existence of characteristic terms for a subtyping relation implies its denotational preciseness.*

This theorem implies the denotational preciseness of the subtypings which are shown to be operationally precise in previous sections. In particular the denotational preciseness of $L_{+\times}^{\to\mu}$ is new, since Ligatti et al. [27] only consider operational preciseness.

7 Conclusion

The present paper discusses some recent results of preciseness for subtyping of typed functional and concurrent calculi.

Operational completeness requires that all empty (i.e. not inhabited) types are less than all inhabited types. This makes unfeasible an operationally complete subtyping for the pure λ-calculus, both in the case of polymorphic types [28] and of intersection and union types. In fact inhabitation is undecidable for polymorphic types being equivalent to derivability in second order logic, while [37] shows undecidability of inhabitation for intersection types, which implies undecidability of inhabitation for intersection and union types.

An interesting open problem we plan to study is an extension of λ-calculus enjoying operational preciseness for the decidable subtypings between polymorphic types discussed in [28,36].

The formulation of preciseness along with the proof methods and techniques described in this paper could be useful to examine other subtypings and calculi. Our future work includes the applications to higher-order processes [29–31], polymorphic types [7,18,19], fair subtypings [33,34] and contract subtyping [3]. 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 [24]. Such a characterisation would give preciseness for the many type systems which are instances of [24]. The notion of subtyping for session types is clearly connected with that of type duality. Various definitions of dualities are compared in [5], and we plan to investigate if completeness of subtyping can be used in finding the largest safe duality.

A last question we plan to investigate is whether preciseness of subtyping is meaningful for object-oriented calculi [1].

References

1. Abadi, M., Cardelli, L.: A Theory of Objects. Monographs in Computer Science. Springer, New York (1996)
2. van Bakel, S., Dezani-Ciancaglini, M., de' Liguoro, U., Motohama, Y.: The minimal relevant logic and the call-by-value lambda calculus. Technical Report TR-ARP-05-2000, The Australian National University (2000)
3. Barbanera, F., de Liguoro, U.: Two notions of sub-behaviour for session-based client/server systems. In: Kutsia, T., Schreiner, W., Fernández, M. (eds.) PPDP, pp. 155–164. ACM (2010)
4. Barendregt, H., Coppo, M., Dezani-Ciancaglini, M.: A filter lambda model and the completeness of type assignment. J. Symbolic Logic **48**(4), 931–940 (1983)
5. Bernardi, G., Dardha, O., Gay, S.J., Kouzapas, D.: On duality relations for session types. In: Maffei, M., Tuosto, E. (eds.) TGC 2014. LNCS, vol. 8902, pp. 51–66. Springer, Heidelberg (2014)
6. Bonsangue, M.M., Rot, J., Ancona, D., de Boer, F.S., Rutten, J.J.M.M.: A coalgebraic foundation for coinductive union types. In: Esparza, J., Fraigniaud, P., Husfeldt, T., Koutsoupias, E. (eds.) ICALP 2014, Part II. LNCS, vol. 8573, pp. 62–73. Springer, Heidelberg (2014)
7. Caires, L., Pérez, J.A., Pfenning, F., Toninho, B.: Behavioral polymorphism and parametricity in session-based communication. In: Felleisen, M., Gardner, P. (eds.) ESOP 2013. LNCS, vol. 7792, pp. 330–349. Springer, Heidelberg (2013)
8. Castagna, G., De Nicola, R., Varacca, D.: Semantic subtyping for the pi-calculus. Theoret. Comput. Sci. **398**(1–3), 217–242 (2008)
9. Castagna, G., Dezani-Ciancaglini, M., Giachino, E., Padovani, L.: Foundations of session types. In: PPDP, pp. 219–230. ACM (2009)
10. Castagna, G., Frisch, A.: A gentle introduction to semantic subtyping. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 30–34. Springer, Heidelberg (2005)
11. Chen, T.-C., Dezani-Ciancaglini, M., Yoshida, N.: On the preciseness of subtyping in session types. In: Chitil, O., King, A., Danvy, O. (eds.) PPDP, pp. 135–146. ACM Press (2014)
12. Coppo, M., Dezani-Ciancaglini, M., Padovani, L., Yoshida, N.: A gentle introduction to multiparty asynchronous session types. In: Bernardo, M., Johnsen, E.B. (eds.) SFM 2015. LNCS, vol. 9104, pp. 146–178. Springer, Switzerland (2015)
13. Demangeon, R., Honda, K.: Full abstraction in a subtyped pi-calculus with linear types. In: Katoen, J.-P., König, B. (eds.) CONCUR 2011. LNCS, vol. 6901, pp. 280–296. Springer, Heidelberg (2011)
14. Dezani-Ciancaglini, M., de 'Liguoro, U., Piperno, A.: A filter model for concurrent lambda-calculus. SIAM J. Comput. **27**(5), 1376–1419 (1998)
15. Dezani-Ciancaglini, M., Ghilezan, S.: Preciseness of subtyping on intersection and union types. In: Dowek, G. (ed.) RTA-TLCA 2014. LNCS, vol. 8560, pp. 194–207. Springer, Heidelberg (2014)
16. Dezani-Ciancaglini, M., Ghilezan, S., Jaksic, S., Pantovic, J., Yoshida, N.: Precise subtyping for synchronous multiparty sessions. In: PLACES, EPTCS 203, pp. 29–44 (2016)
17. Frisch, A., Castagna, G., Benzaken, V.: Semantic subtyping: dealing set-theoretically with function, union, intersection, and negation types. J. ACM **55**(4), 1–64 (2008)

18. Gay, S.J.: Bounded polymorphism in session types. Math. Struct. Comput. Sci. **18**(5), 895–930 (2008)
19. Goto, M., Jagadeesan, R., Jeffrey, A., Pitcher, C., Riely, J.: An extensible approach to session polymorphism. Math. Struct. Comput. Sci. **26**(3), 465–509 (2016)
20. Harper, R.: Practical Foundations for Programming Languages. Cambridge University Press, Cambridge (2013)
21. Hindley, J.R.: The completeness theorem for typing lambda-terms. Theoret. Comput. Sci. **22**, 1–17 (1983)
22. Honda, K., Vasconcelos, V.T., Kubo, M.: Language primitives and type discipline for structured communication-based programming. In: Hankin, C. (ed.) ESOP 1998. LNCS, vol. 1381, pp. 122–138. Springer, Heidelberg (1998)
23. Honda, K., Yoshida, N., Carbone, M.: Multiparty asynchronous session types. In POPL, pp. 273–284. ACM (2008). A full version will appear in Journal of the Association for Computing Machinery
24. Igarashi, A., Kobayashi, N.: A generic type system for the pi-calculus. Theoret. Comput. Sci. **311**(1–3), 121–163 (2004)
25. Ishihara, H., Kurata, T.: Completeness of intersection and union type assignment systems for call-by-value lambda-models. Theoret. Comput. Sci. **272**(1–2), 197–221 (2002)
26. Kouzapas, D., Yoshida, N.: Globally governed session semantics. In: D'Argenio, P.R., Melgratti, H. (eds.) CONCUR 2013. LNCS, vol. 8052, pp. 395–409. Springer, Heidelberg (2013)
27. Ligatti, J., Blackburn, J., Nachtigal, M.: On subtyping-relation completeness, with an application to iso-recursive types. Technical report, University of South Florida (2014)
28. Mitchell, J.C.: Polymorphic type inference and containment. Inf. Comput. **76**(2/3), 211–249 (1988)
29. Mostrous, D.: Session Types in Concurrent Calculi: Higher-Order Processes and Objects. PhD thesis, Imperial College London (2009)
30. Mostrous, D., Yoshida, N.: Session-based communication optimisation for higher-order mobile processes. In: Curien, P.-L. (ed.) TLCA 2009. LNCS, vol. 5608, pp. 203–218. Springer, Heidelberg (2009)
31. Mostrous, D., Yoshida, N.: Session typing and asynchronous subtyping for the higher-order π-calculus. Inf. Comput. **241**, 227–263 (2015)
32. Mostrous, D., Yoshida, N., Honda, K.: Global principal typing in partially commutative asynchronous sessions. In: Castagna, G. (ed.) ESOP 2009. LNCS, vol. 5502, pp. 316–332. Springer, Heidelberg (2009)
33. Padovani, L.: Fair subtyping for multi-party session types. In: De Meuter, W., Roman, G.-C. (eds.) COORDINATION 2011. LNCS, vol. 6721, pp. 127–141. Springer, Heidelberg (2011)
34. Padovani, L.: Fair subtyping for open session types. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M., Peleg, D. (eds.) ICALP 2013, Part II. LNCS, vol. 7966, pp. 373–384. Springer, Heidelberg (2013)
35. Pierce, B.C.: Types and Programming Languages. MIT Press, Cambridge (2002)
36. Tiuryn, J., Urzyczyn, P.: The subtyping problem for second-order types is undecidable. Inf. Comput. **179**(1), 1–18 (2002)
37. Urzyczyn, P.: The emptiness problem for intersection types. J. Symbolic Logic **64**(3), 1195–1215 (1999)
38. Vouillon, J.: Subtyping union types. In: Marcinkowski, J., Tarlecki, A. (eds.) CSL 2004. LNCS, vol. 3210, pp. 415–429. Springer, Heidelberg (2004)