Acta Informatica manuscript No.
(will be inserted by the editor)

Subtyping for Session Types in the Pi Calculus

Simon Gay[1], **Malcolm Hole**[2⋆]

[1] Department of Computing Science, University of Glasgow, UK
[2] Department of Computer Science, Royal Holloway, University of London, UK

Received: date / Revised version: date

Abstract. Extending the pi calculus with the *session types* proposed by Honda *et al.* allows high-level specifications of structured patterns of communication, such as client-server protocols, to be expressed as types and verified by static typechecking. We define a notion of subtyping for session types, which allows protocol specifications to be extended in order to describe richer behaviour; for example, an implemented server can be refined without invalidating type-correctness of an overall system. We formalize the syntax, operational semantics and typing rules of an extended pi calculus, prove that typability guarantees absence of run-time communication errors, and show that the typing rules can be transformed into a practical typechecking algorithm.

1 Introduction

The pi calculus [21,30] has been used as a vehicle for much research on static type systems for concurrent programming languages, for example [6,16–18,20,25,27,28,32,34], in addition to its widespread use for modelling and reasoning about concurrent systems. In such systems, successfully typechecking a program guarantees that certain kinds of error do not occur at run-time. The eliminated errors range from disagreements between sender and receiver about the expected type of a message [20] to deadlocks [17].

⋆ Malcolm Hole died on 28th February 2004, a few weeks after the original submission of this paper.

Because the pi calculus is based on point-to-point communication on named channels, a central idea of many of these type systems is that each channel has a type which specifies, uniformly, the type of message which it can carry: for example, every message on a channel of type $\hat{}[\mathsf{int}]$ consists of an integer value.

In distributed systems it is common for communication between two processes to consist of a structured dialogue described by a protocol, which specifies the format and direction of each message in a sequence. This view of structured communication does not map well onto a type system which requires each channel to carry messages of just one type. To address this problem, Honda *et al.* [14, 15,31] introduced *session types*, which allow non-uniform but structured sequences of messages to be specified. For example, the type $?[\mathsf{int}].![\mathsf{bool}].\mathsf{end}$ describes a channel which can be used to receive an integer, then send a boolean, and then must not be used again. The main contribution of the present paper is to define a notion of subtyping for session types. This increases the flexibility of the type system by allowing the participants in a dialogue to follow different protocols which are nevertheless compatible in a sense defined by the subtyping relation. The definition of subtyping for session types first appeared in our earlier paper [7]; the present paper is substantially revised, extended and corrected. We also show that the theoretical presentation of the type system can be converted into a practical typechecking algorithm; this is a new result of the present paper.

In Section 2 we describe session types informally, explain why subtyping is useful, and discuss related work. In Section 3 we formally define the syntax, operational semantics, and type system of our version of the pi calculus, and prove that every process which is typable in the simply-typed pi calculus is also typable in our system. In Section 4 we prove that typability in our system is preserved by the reductions of the operational semantics, which implies that typable processes have desirable run-time behaviour in a sense which we make precise. Up to this point, the paper is a revised and expanded version of our earlier conference paper [7] and technical report [8]. In Section 5 we present substantial new results by showing that the typing rules of our system, presented declaratively in Section 3 for the sake of formal convenience, can be transformed into a practical typechecking algorithm; we prove that this algorithm is sound with respect to the original typing rules. Finally, Section 6 concludes.

2 Session Types and Subtyping

2.1 A Client-Server System and its Session Types

Consider a server for mathematical operations which provides two services: addition of integers, yielding an integer result, and testing of integers for equality, yielding a boolean result. In use, the server is executed in parallel with a client, and they communicate on a channel x. The client chooses a service by sending either plus or eq on x, then sends the arguments of the chosen operation, and finally receives the result. This protocol is specified formally by the session type S, which describes the type of x from the server's viewpoint.

$$S = \&\langle\, \mathsf{plus}\colon ?[\mathsf{int}].?[\mathsf{int}].![\mathsf{int}].\mathsf{end},\\ \mathsf{eq}\colon ?[\mathsf{int}].?[\mathsf{int}].![\mathsf{bool}].\mathsf{end}\,\rangle$$

The $\&\langle \ldots \rangle$ constructor specifies that a choice is offered between, in this case, two options, labelled plus and eq. Each label leads to a type describing the subsequent pattern of communication, different in each case. The constructors $?[\ldots]$ and $![\ldots]$ indicate receiving and sending, respectively. Sequencing is indicated by $.$ and end marks the end of the interaction.

The server (parameterized on the channel x, for later use) can be implemented by

$$\mathsf{serverbody}(x) = x\triangleright\{\, \mathsf{plus}\colon x?[u{:}\mathsf{int}].x?[v{:}\mathsf{int}].x![u+v].\mathbf{0},\\ \mathsf{eq}\colon x?[u{:}\mathsf{int}].x?[v{:}\mathsf{int}].x![u=v].\mathbf{0}\,\}.$$

Here $x\triangleright\{\ldots\}$ is the branching construct, corresponding to the type constructor $\&\langle \ldots \rangle$; it receives a label and executes the appropriate code. The rest of the syntax is pi calculus extended with arithmetic operations: $x?[u{:}\mathsf{int}].P$ receives an integer value on channel x, binds it to the name u, and then executes P; $x![u+v].Q$ sends the value of the expression $u+v$ on channel x and then executes Q; $\mathbf{0}$ is the terminated process.

Our type system, defined formally in Section 3, consists of inference rules for judgements of the form $\Gamma \vdash P$, where Γ is an environment of typed channel names and P is a process. We can formally derive

$$x{:}S \vdash \mathsf{serverbody}(x)$$

and in this case it is also easy to see, informally, that the structure of the communications in server exactly matches the structure of S.

A client must be typechecked against the dual type

$$\overline{S} = \oplus\langle\, \mathsf{plus}\colon ![\mathsf{int}].![\mathsf{int}].?[\mathsf{int}].\mathsf{end}, \mathsf{eq}\colon ![\mathsf{int}].![\mathsf{int}].?[\mathsf{bool}].\mathsf{end}\,\rangle$$

in which the $\oplus\langle \ldots \rangle$ constructor specifies the range of options from which a choice must be made. Each label is followed by a type which describes the subsequent interaction, and the pattern of sending and receiving is the opposite of the pattern which appears in the type of the server. We imagine the type $\overline{S}$ being published as part of the specification of the server, so that client implementors can typecheck their code.

A possible client, also parameterized on the channel x, is

$$\mathsf{clientbody}(x) = x \triangleleft \mathsf{plus}.x!\![2].x!\![3].x?[u{:}\mathsf{int}].\mathbf{0}$$

where $x \triangleleft \mathsf{plus}$ sends the label which indicates the desired service. More realistically, $\mathbf{0}$ would be replaced by some process which used the received value u. Our type system allows us to derive

$$x{:}\overline{S} \vdash \mathsf{clientbody}(x).$$

Explicitly representing the server's end of the channel by x^+ and the client's end by x^-, the operational semantics presented in Section 3 defines the behaviour of the parallel combination

$$\mathsf{clientbody}(x^-) \mid \mathsf{serverbody}(x^+)$$

as a sequence of reduction steps, each step corresponding to a communication.

In a complete system, there must be a way for the client and server to establish their connection along channel x. Honda *et al.* treat session channels separately from standard pi calculus channels, creating them with matching request and accept constructs in the client and server respectively; initially [31] they did not allow session channels to be transmitted between processes, and later [15] they introduced throw and catch constructs for sending and receiving session channels. We prefer to control session channels through the type system rather than with special syntax, so we allow them to be created using the standard pi calculus ν construct and transmitted just like standard channels.

We structure a complete client-server system as follows. The client creates a session channel x of type S and sends one end of it to the server along a standard channel a of type $\widehat{\ }[S]$. The system is defined and typechecked as follows.

$$\begin{aligned} \mathsf{server} &= a?[y{:}S].\mathsf{serverbody}(y) \\ \mathsf{client} &= (\nu x : S)(a![x^+].\mathsf{clientbody}(x^-)) \end{aligned}$$

$$a{:}\widehat{\ }[S] \vdash \mathsf{client} \mid \mathsf{server}$$

The system reduces to

$$(\nu x : S)(\mathsf{clientbody}(x^-) \mid \mathsf{serverbody}(x^+))$$

by communication on a and standard pi calculus scope extrusion, resulting in a private connection between client and server. Using standard pi calculus programming techniques, it is straightforward to modify this implementation to produce a multi-threaded server which can be accessed via the channel a by any number of clients, each obtaining a private session with a separate thread.

Each end (x^+ or x^-) of a session channel x must be owned by just one process at any time, because otherwise typechecking individual processes would not guarantee safety of the complete system. For example, allowing the typing

$$x^+{:}?[\mathsf{int}].?[\mathsf{bool}].\mathsf{end}, x^-{:}![\mathsf{int}].![\mathsf{bool}].\mathsf{end} \vdash \\ x^-![2].x^-![\mathit{true}].\mathbf{0} \mid x^-![3].x^-![\mathit{false}].\mathbf{0} \mid x^+?[u{:}\mathsf{int}].x^+?[v{:}\mathsf{bool}].\mathbf{0}$$

is not safe because the process can reduce to

$$x^-![\mathit{true}].\mathbf{0} \mid x^-![3].x^-![\mathit{false}].\mathbf{0} \mid x^+?[v{:}\mathsf{bool}].\mathbf{0}$$

and now an incorrect communication is possible, substituting the integer 3 for the boolean variable v.

The type system must maintain unique ownership in the presence of channel mobility: for example, after sending x^+ to server, client must not use x^+ for communication. This restriction is related to *linear* [9] control of values and our type system treats session channels in a similar way to the linear and *linearized* channels studied by Kobayashi, Pierce and Turner [18]. The difference is that each end of a session channel may be used many times by the process which owns it.

2.2 Upgrading the Server and the Need for Subtypes

Suppose that the server is extended in two ways: by adding a negation service neg, and by extending the equality test service to real numbers. The corresponding session type is

$$\begin{aligned} T = \&\langle\ & \mathsf{plus}{:}?[\mathsf{int}].?[\mathsf{int}].![\mathsf{int}].\mathsf{end}, \\ & \mathsf{eq}{:}?[\mathsf{real}].?[\mathsf{real}].![\mathsf{bool}].\mathsf{end}, \\ & \mathsf{neg}{:}?[\mathsf{int}].![\mathsf{int}].\mathsf{end}\ \rangle. \end{aligned}$$

With a suitable server implementation we now have

$$a{:}\widehat{\ }[T] \vdash \mathsf{newserver}$$

and the original typing of the client

$$a{:}\widehat{\ }[S] \vdash \mathsf{client}$$

means that the system is no longer typable, because typing $\mathsf{client} \mid \mathsf{server}$ requires a to have the same type throughout the system. However, if we assume that int is a subtype of real ($\mathsf{int} \leqslant \mathsf{real}$) in the underlying data language, it is clear that the system still executes without communication errors.

By introducing a suitable notion of subtyping for session types we can account for this situation. The principle underlying the definition is that if $S_1 \leqslant S_2$ then a session channel of type S_1 can safely be used anywhere that a session channel of type S_2 is expected. Technically, if $x{:}S_2 \vdash P$ and $S_1 \leqslant S_2$ then $x{:}S_1 \vdash P$. The definition of subtyping is presented in Section 3. In summary, it is possible to move up the subtype relation by changing message types covariantly in input positions or contravariantly in output positions; changing the label-sets covariantly in branch types or contravariantly in choice types; or changing the continuation types covariantly in input, output, branch and choice types. The variance of input and output, and analogously of the label-sets in branch and choice, are the same as in Pierce and Sangiorgi's [25] system of input/output subtyping; the uniform covariance of the continuation types is slightly counterintuitive.

For the mathematical server, this means that $S \leqslant T$. It is safe for the new server to communicate on a channel of type S: this just means that the neg service is never used and the eq service is only used with integer values.

When sending, the type of the actual message is allowed to be a subtype of the message type specified by the channel type, because it is safe for the receiver to be given a value whose type is a subtype of the expected type. This means that from

$$a{:}\widehat{\ }[T], x^{-}{:}\overline{S} \vdash \mathsf{clientbody}(x^{-})$$

we can derive

$$a{:}\widehat{\ }[T], x^{+}{:}S, x^{-}{:}\overline{S} \vdash a![x^{+}].\mathsf{clientbody}(x^{-})$$

and hence

$$a{:}\widehat{\ }[T] \vdash \mathsf{client}$$

which is compatible with the new server's publication of $a{:}\widehat{\ }[T]$ as its access channel. A complete typing derivation for this client-server system appears in Section 3.5.

We have presented a larger example, based on the POP3 protocol [22], elsewhere [7, 8].

2.3 Related Work

Session types were originally formulated by Honda [14] and developed further by Takeuchi, Honda and Kubo [31] and Honda, Vasconcelos and Kubo [15]. The key difference between our formulation of the core language (without subtyping) and theirs is that whereas they use special syntax for creation (`request`, `accept`) and transmission (`throw`, `catch`) of sessions, we just use the standard pi calculus primitives for channel creation and transmission, and handle all of the special behaviour of session channels through the type system. Bonelli, Compagnoni and Gunter [1,2] have extended session types with a system for checking correspondence assertions, in the style of Gordon and Jeffrey [10], in order to specify more detailed properties of protocols.

Subtyping for session types was first defined by the present authors [7,8] and the present paper is intended to be the definitive account of our type system. Several authors have used our definition of subtyping as the basis for further developments and applications of session types. Vallecillo, Vasconcelos and Ravara [33] have used session types to specify component interfaces in CORBA. They use a notion of *compatibility* between types, which is derived from our notion of subtyping: a client type T and a server type U are compatible, $T \bowtie U$, if and only if $\overline{T} \leqslant U$. Neubauer and Thiemann [24] use a type system based on session types with subtyping and singleton types to describe specialization of protocols by elimination of unnecessary behaviour.

Igarashi and Kobayashi [16] have developed a generic framework in which a range of type systems can be defined for the pi calculus. Although able to express sequencing of input and output types in a similar way to session types, it appears not to be able to express branching types because they require a dependency between the label and the type describing the subseqent behaviour.

Rajamani *et al.*'s *Behave* [3,29] uses CCS to describe properties of pi calculus programs, verified via a combination of typechecking and model checking. Although *Behave* addresses a wider range of properties, we expect session types to be easier to implement, because only typechecking is involved, and well suited to a programming environment, because of the use of familiar concepts of typing and subtyping.

Some recent work has begun to transfer session types from languages based on pi calculus to standard programming paradigms. Vasconcelos, the first author and Ravara [35] have defined a system of session types for a functional language with side-effecting input/output operations. Dezani-Ciancaglini *et al.* [5] have defined a distributed object-oriented language with session types. Neubauer

and Thiemann [23] have encoded a version of session types in Haskell, and proved that the embedding preserves typings.

Some type-theoretic approaches to verification of resource-access protocols have more general aims than session types. However, in the particular case of communication channels, session types give more detailed specifications of correct behaviour. *Cyclone* [12] is a C-like type-safe polymorphic imperative language. It features region-based memory management, and more recently threads and locks [11], via a sophisticated type system. The type system guarantees that locks are acquired and released in a correct sequence, but does not specify a protocol for using an object once the lock has been acquired. In the *Vault* system [4] annotations are added to C programs, in order to describe protocols that a compiler can statically enforce; these protocols specify the permitted state transition sequences of tracked run-time objects. We can regard a session type as a specification of state transitions for a channel, which is similar to a specification in *Vault*, but in addition, session types specify the types of individual messages.

3 The Language: Syntax, Semantics, Type System

Our language is based on the polyadic pi calculus of Milner *et al.* [20,21,30]. We add the constructs proposed by Honda *et al.* [14,15, 31] which allow external choice, between a collection of labelled processes, to be resolved by transmission of a label on a channel. We omit internal choice and matching of names: these features have little interaction with the type system and can easily be added if desired. The type system is based on our formulation [7] of the session types of Honda *et al.* [14,15,31]. It incorporates our notion of subtyping for session types [7,8].

To simplify the presentation we have restricted the language to a pure calculus of names and channel types. It is straightforward to add data types and data expressions, as required by the examples in Section 2; for example, we have incorporated a boolean type in an earlier presentation [8].

3.1 Syntax

The syntax of types is defined by the grammar in Figure 1, assuming an infinite collection $X, Y \ldots$ of type variables and an infinite collection $l_1, l_2, \ldots$ of *labels*. We often write $\widetilde{T}$ for a sequence $T_1, \ldots, T_n$ of types, and $\widetilde{l} : \widetilde{T}$ for a sequence $l_1 : T_1, \ldots, l_n : T_n$ of labelled types.

Session types	$S ::= X$		*type variable*
	$\mid$	end	*terminated session*
	$\mid$	$?[T_1, \ldots, T_n].S$	*input*
	$\mid$	$![T_1, \ldots, T_n].S$	*output*
	$\mid$	$\&\langle\, l_1 : S_1, \ldots, l_n : S_n \,\rangle$	*branch*
	$\mid$	$\oplus\langle\, l_1 : S_1, \ldots, l_n : S_n \,\rangle$	*choice*
	$\mid$	$\mu X.S$	*recursive session type*
Types	$T ::= X$		*type variable*
	$\mid$	S	*session type*
	$\mid$	$\widehat{\ }[T_1, \ldots, T_n]$	*standard channel type*
	$\mid$	$\mu X.T$	*recursive channel type*

Fig. 1 Syntax of types

Recursive types are required to be *contractive*, containing no subexpressions of the form $\mu X.\mu X_1 \ldots \mu X_n.X$. Each session type S has a *dual* type $\overline{S}$, defined recursively by the equations in Figure 2.

The syntax of processes is defined by the grammar in Figure 3. We assume an infinite collection of *names* $x, y, z, \ldots$, which is disjoint from the set of labels. Names may be *polarized*, occurring as x^+ or x^- or simply as x. We write x^p for a general polarized name, where p represents an optional polarity. We often write $\widetilde{x^p}$ for a sequence $x_1^{p_1}, \ldots, x_n^{p_n}$ of polarized names. Duality on polarities, written $\overline{p}$, exchanges $^+$ and $^-$. As is common in presentations of the pi calculus, we do not distinguish between names and variables. The definitions of binding and the *free names* of a process, $fn(P)$, are slightly nonstandard. Binding occurrences of names are x in $(\nu x{:}T)P$ and $\widetilde{y}$ in $x^p?[\widetilde{y}{:}\widetilde{T}].P$. In $(\nu x{:}T)P$, both x^+ and x^- may occur in P, and both are bound. In $x^p?[\widetilde{y}{:}\widetilde{T}].P$, for each i, only y_i (unpolarized) may occur in P. This will become clear when the type system is presented, in Section 3.4.

To facilitate the presentation of the type system, binding occurrences of names are annotated with types. We work up to α-equivalence as usual, and in proofs we assume that all bound names are distinct from each other and from all free names.

3.2 Operational Semantics

Following one of the standard approaches to pi calculus semantics [20] we define a reduction relation on processes, making use of a structural congruence relation. Structural congruence is the smallest congruence relation on processes which contains α-equivalence and

$$
\begin{aligned}
\overline{X} &= X \\
\overline{\mathsf{end}} &= \mathsf{end} \\
\overline{?[T_1,\ldots,T_n].S} &= ![T_1,\ldots,T_n].\overline{S} \\
\overline{![T_1,\ldots,T_n].S} &= ?[T_1,\ldots,T_n].\overline{S} \\
\overline{\&\langle\, l_1 : S_1, \ldots, l_n : S_n \,\rangle} &= \oplus\langle\, l_1 : \overline{S_1}, \ldots, l_n : \overline{S_n} \,\rangle \\
\overline{\oplus\langle\, l_1 : S_1, \ldots, l_n : S_n \,\rangle} &= \&\langle\, l_1 : \overline{S_1}, \ldots, l_n : \overline{S_n} \,\rangle \\
\overline{\mu X.S} &= \mu X.\overline{S}
\end{aligned}
$$

Fig. 2 The dual of a session type

$P, Q ::= \mathbf{0}$		*terminated process*
	$P \mid Q$	*parallel combination*
	$!P$	*replication*
	$x^p?[y_1 : T_1, \ldots, y_n : T_n].P$	*input*
	$x^p![y_1{}^{p_1}, \ldots, y_n{}^{p_n}].P$	*output*
	$(\nu x{:}T)P$	*channel creation*
	$x^p \triangleright \{l_1 : P_1, \ldots, l_n : P_n\}$	*branch*
	$x^p \triangleleft l.P$	*choice*

Fig. 3 Syntax of processes

$$
\begin{aligned}
P \mid \mathbf{0} &\equiv P && \text{S-Unit} \\
P \mid Q &\equiv Q \mid P && \text{S-Comm} \\
P \mid (Q \mid R) &\equiv (P \mid Q) \mid R && \text{S-Assoc} \\
!P &\equiv P \mid !P && \text{S-Rep} \\
(\nu x{:}T)P \mid Q &\equiv (\nu x{:}T)(P \mid Q) \text{ if } x, x^+, x^- \notin \mathit{fn}(Q) \text{ and } T \neq \mathsf{end} && \text{S-Extr} \\
(\nu x{:}T)\mathbf{0} &\equiv \mathbf{0} \text{ if } T \text{ is not a session type} && \text{S-Nil} \\
(\nu x{:}\mathsf{end})\mathbf{0} &\equiv \mathbf{0} && \text{S-NilS} \\
(\nu x{:}T)(\nu y{:}U)P &\equiv (\nu y{:}U)(\nu x{:}T)P && \text{S-Switch}
\end{aligned}
$$

Fig. 4 Structural congruence

is closed under the equations in Figure 4. The structural congruence rules are standard. Rule S-NilS specifies the type end in the ν-binding, because of the way in which the type system (Section 3.4) requires the $\mathbf{0}$ process to be typed in an environment of fully-used channels.

The reduction relation is defined inductively by the rules in Figure 5. To enable our Type Preservation Theorem to be stated (Theorem 1, Section 4), reductions are annotated with labels of the form α, l. These labels indicate the channel name and branch selection label, if any, which are involved in each reduction. Consider a reduction which involves communication on channel x. If x is not ν-bound then $\alpha = x$. If x is ν-bound then $\alpha = \tau$. If the reduction consists of trans-

$$x^p?[\widetilde{y}{:}\widetilde{T}].P \mid x^{\bar{p}}![\widetilde{z}^{\tilde{q}}].Q \xrightarrow{x,\_} P\{\widetilde{z}^{\tilde{q}}/\widetilde{y}\} \mid Q \qquad \text{R-Com}$$

$$\frac{p \text{ is either } + \text{ or } - \qquad 1 \leqslant i \leqslant n}{x^p \triangleright \{l_1 : P_1, \ldots, l_n : P_n\} \mid x^{\bar{p}} \triangleleft l_i.Q \xrightarrow{x,l_i} P_i \mid Q} \text{R-Select}$$

$$\frac{P \xrightarrow{\alpha,l} P' \qquad \alpha \neq x \qquad T \text{ is not a session type}}{(\nu x : T)P \xrightarrow{\alpha,l} (\nu x : T)P'} \text{R-New}$$

$$\frac{P \xrightarrow{x,l} P'}{(\nu x : S)P \xrightarrow{\tau,\_} (\nu x : tail(S,l))P'} \text{R-NewS}$$

$$\frac{P \xrightarrow{\alpha,l} P'}{P \mid Q \xrightarrow{\alpha,l} P' \mid Q} \text{R-Par} \qquad \frac{P' \equiv P \quad P \xrightarrow{\alpha,l} Q \quad Q \equiv Q'}{P' \xrightarrow{\alpha,l} Q'} \text{R-Cong}$$

Fig. 5 The reduction relation

$$\begin{aligned}
x^q\{\widetilde{u}^{\tilde{p}}/\widetilde{v}\} &= x^q \quad \text{if } x \notin \widetilde{v} \\
x\{\widetilde{u}^{\tilde{p}}/\widetilde{v}\} &= u_i^{p_i} \quad \text{if } x = v_i \\
\mathbf{0}\{\widetilde{u}^{\tilde{p}}/\widetilde{v}\} &= \mathbf{0} \\
(P \mid Q)\{\widetilde{u}^{\tilde{p}}/\widetilde{v}\} &= P\{\widetilde{u}^{\tilde{p}}/\widetilde{v}\} \mid Q\{\widetilde{u}^{\tilde{p}}/\widetilde{v}\} \\
(!P)\{\widetilde{u}^{\tilde{p}}/\widetilde{v}\} &= !(P\{\widetilde{u}^{\tilde{p}}/\widetilde{v}\}) \\
(x^q?[\widetilde{y}{:}\widetilde{T}].P)\{\widetilde{u}^{\tilde{p}}/\widetilde{v}\} &= x^q\{\widetilde{u}^{\tilde{p}}/\widetilde{v}\}?[\widetilde{y}{:}\widetilde{T}].P\{\widetilde{u}^{\tilde{p}}/\widetilde{v}\} \\
(x^q![\widetilde{y}].P)\{\widetilde{u}^{\tilde{p}}/\widetilde{v}\} &= x^q\{\widetilde{u}^{\tilde{p}}/\widetilde{v}\}![\widetilde{y}\{\widetilde{u}^{\tilde{p}}/\widetilde{v}\}].P\{\widetilde{u}^{\tilde{p}}/\widetilde{v}\} \\
((\nu x{:}T)P)\{\widetilde{u}^{\tilde{p}}/\widetilde{v}\} &= (\nu x{:}T)P\{\widetilde{u}^{\tilde{p}}/\widetilde{v}\} \\
(x^q \triangleright \{\widetilde{l} : \widetilde{P}\})\{\widetilde{u}^{\tilde{p}}/\widetilde{v}\} &= x^q\{\widetilde{u}^{\tilde{p}}/\widetilde{v}\} \triangleright \{\widetilde{l} : \widetilde{P}\{\widetilde{u}^{\tilde{p}}/\widetilde{v}\}\} \\
(x^q \triangleleft l.P)\{\widetilde{u}^{\tilde{p}}/\widetilde{v}\} &= x^q\{\widetilde{u}^{\tilde{p}}/\widetilde{v}\} \triangleleft l.P\{\widetilde{u}^{\tilde{p}}/\widetilde{v}\}
\end{aligned}$$

Fig. 6 Substitution

$$\begin{aligned}
tail(?[\widetilde{T}].S, _) &= S \\
tail(![\widetilde{T}].S, _) &= S \\
tail(\&\langle \widetilde{l} : \widetilde{S} \rangle, l_i) &= S_i \\
tail(\oplus\langle \widetilde{l} : \widetilde{S} \rangle, l_i) &= S_i \\
tail(\mu X.S, l) &= tail(S\{\mu X.S/X\}, l)
\end{aligned}$$

Fig. 7 The *tail* function

mission of a choice label then l is that label, otherwise $l = \_$. We assume that the label $\_$ does not occur as a choice label.

Rule R-Com is the standard communication reduction for the pi calculus. Substitution of polarized names for unpolarized variables is defined in Figure 6; it is only necessary to substitute for input-bound variables, and these are not polarized. The channel on which communication takes place, and the names which are transmitted, are all polarized, perhaps as the result of substitutions arising from earlier reductions.

Rule R-Select resolves a choice between labelled processes by sending a label along a channel. The standard rule for reduction under a ν-binding is replaced by two rules, R-New and R-NewS. These rules use the annotation on the reduction in the hypothesis to calculate the correct type for the ν-binding in the conclusion. The function *tail* is defined in Figure 7. Rules R-Par and R-Cong are standard.

3.3 Subtyping

The inductive rules in Figure 8, which define subtyping between non-recursive types, show the key features of subtyping for session types. If T is a subtype of U, written $T \leqslant U$, then a channel of type T may safely be used wherever a channel of type U is expected. The definition of subtyping for session types, and especially recursive session types, is one of the main contributions of the paper.

Rules S-InS and S-OutS specify covariance and contravariance, respectively, in the message type, and covariance in the continuation type. Rule S-Chan specifies invariance in the message type. The variance of the message type in these cases is the same as in Pierce and Sangiorgi's system of input/output subtyping in the pi calculus [25].

Rules S-Branch and S-Choice specify that branch is covariant, and choice contravariant, in the set of labels. This is as expected if a branch is viewed as an input and a choice as an output. Less intuitive is the fact that both branch and choice are covariant in the continuation types, but the definition is justified by the proof of safety of the type system.

To extend subtyping to recursive types, we use a coinductive definition.

Definition 1. *For all types T, define $unfold(T)$ by recursion on the structure of T:*

$$unfold(\mu X.T) = unfold(T\{\mu X.T/X\})$$

and in all other cases, $unfold(T) = T$.

Because we assume that recursive types are contractive, *unfold* terminates. For any recursive type T, $\mathit{unfold}(T)$ is the result of repeatedly unfolding the top level recursion until a non-recursive type constructor is reached.

Definition 2. *Let Type be the set of all closed type expressions defined by the grammar for T (Figure 1).*

Definition 3. *A relation $R \subseteq \mathit{Type} \times \mathit{Type}$ is a* type simulation *if $(T, U) \in R$ implies the following conditions:*

1. *If $\mathit{unfold}(T) = \hat{}[T_1, \ldots, T_n]$ then $\mathit{unfold}(U) = \hat{}[U_1, \ldots, U_n]$ and for all $i \in \{1, \ldots, n\}$, $(T_i, U_i) \in R$ and $(U_i, T_i) \in R$.*
2. *If $\mathit{unfold}(T) = ?[T_1, \ldots, T_n].S_1$ then $\mathit{unfold}(U) = ?[U_1, \ldots, U_n].S_2$ and $(S_1, S_2) \in R$ and for all $i \in \{1, \ldots, n\}$, $(T_i, U_i) \in R$.*
3. *If $\mathit{unfold}(T) = ![T_1, \ldots, T_n].S_1$ then $\mathit{unfold}(U) = ![U_1, \ldots, U_n].S_2$ and $(S_1, S_2) \in R$ and for all $i \in \{1, \ldots, n\}$, $(U_i, T_i) \in R$.*
4. *If $\mathit{unfold}(T) = \&\langle\, l_1 : S_1, \ldots, l_m : S_m \,\rangle$ then $\mathit{unfold}(U) = \&\langle\, l_1 : S'_1, \ldots, l_n : S'_n \,\rangle$ and $m \leqslant n$ and for all $i \in \{1, \ldots, m\}$, $(S_i, S'_i) \in R$.*
5. *If $\mathit{unfold}(T) = \oplus\langle\, l_1 : S_1, \ldots, l_m : S_m \,\rangle$ then $\mathit{unfold}(U) = \oplus\langle\, l_1 : S'_1, \ldots, l_n : S'_n \,\rangle$ and $n \leqslant m$ and for all $i \in \{1, \ldots, n\}$, $(S_i, S'_i) \in R$.*
6. *If $\mathit{unfold}(T) = \mathsf{end}$ then $\mathit{unfold}(U) = \mathsf{end}$.*

Definition 4. *The* coinductive subtyping relation $\leqslant_c$ *is defined by $T \leqslant_c U$ if and only if there exists a type simulation R such that $(T, U) \in R$.*

Definition 5. *If $\widetilde{T} = T_1, \ldots, T_n$ and $\widetilde{U} = U_1, \ldots, U_n$ and for all $i \in \{1, \ldots, n\}$, $(T_i \leqslant_c U_i)$, then we write $\widetilde{T} \leqslant_c \widetilde{U}$.*

In a practical language we would want to regard the branches in $\&\langle\, l_i : T_i \,\rangle_{1 \leqslant i \leqslant n}$ and $\oplus\langle\, l_i : T_i \,\rangle_{1 \leqslant i \leqslant n}$ as functions from labels to types, not as sequences; this would allow the subtyping relation to vary the order of the branches. We do not discuss this point further in the present paper.

3.4 Type System

The rules in Figure 9 inductively define judgements of the form $\Gamma \vdash P$ where Γ is an *environment*. Such a judgement means that the process P uses channels as specified by the types in Γ. A process is either correctly typed or not; we do not assign types to processes.

$$\frac{}{\mathsf{end} \leqslant \mathsf{end}}\ \text{S-End} \qquad \frac{\forall i.(T_i \leqslant U_i) \qquad \forall i.(U_i \leqslant T_i)}{\widehat{\ }[\widetilde{T}] \leqslant \widehat{\ }[\widetilde{U}]}\ \text{S-Chan}$$

$$\frac{\forall i.(T_i \leqslant U_i) \qquad V \leqslant W}{?[\widetilde{T}].V \leqslant ?[\widetilde{U}].W}\ \text{S-InS} \qquad \frac{\forall i.(U_i \leqslant T_i) \qquad V \leqslant W}{![\widetilde{T}].V \leqslant ![\widetilde{U}].W}\ \text{S-OutS}$$

$$\frac{m \leqslant n \qquad \forall i \in \{1,\ldots,m\}.S_i \leqslant T_i}{\&\langle\, l_i : S_i \,\rangle_{1\leqslant i\leqslant m} \leqslant \&\langle\, l_i : T_i \,\rangle_{1\leqslant i\leqslant n}}\ \text{S-Branch}$$

$$\frac{m \leqslant n \qquad \forall i \in \{1,\ldots,m\}.S_i \leqslant T_i}{\oplus\langle\, l_i : S_i \,\rangle_{1\leqslant i\leqslant n} \leqslant \oplus\langle\, l_i : T_i \,\rangle_{1\leqslant i\leqslant m}}\ \text{S-Choice}$$

Fig. 8 Subtyping for non-recursive types

Definition 6. *An environment Γ is a function from optionally polarized names to types. If $x^p \in dom(\Gamma)$ and $\Gamma(x^p) = T$ then we write $x^p : T \in \Gamma$. Similarly, we sometimes write an environment explicitly as $\Gamma = x_1^{p_1} : T_1, \ldots, x_n^{p_n} : T_n$. If $x^p \notin dom(\Gamma)$ then we write $\Gamma, x^p : T$ for the environment which extends Γ by mapping x^p to T, as long as this environment satisfies the condition below.*

For any environment Γ and any name x, exactly one of the following conditions must apply.

1. *$x \notin dom(\Gamma)$ and $x^+ \notin dom(\Gamma)$ and $x^- \notin dom(\Gamma)$.*
2. *$\Gamma(x) = T$ and $x^+ \notin dom(\Gamma)$ and $x^- \notin dom(\Gamma)$.*
3. *$\Gamma(x^+) = S$ and $x \notin dom(\Gamma)$ and $x^- \notin dom(\Gamma)$.*
4. *$\Gamma(x^-) = S$ and $x \notin dom(\Gamma)$ and $x^+ \notin dom(\Gamma)$.*
5. *$\Gamma(x^+) = S$ and $\Gamma(x^-) = S'$ and $x \notin dom(\Gamma)$.*

At several points in the definition of the type system, we need to include the condition that two session types S and S' are dual. In the presence of recursive types, it is not sufficient to specify that $S' = \overline{S}$ [33]. For example, we want $\mu X.?[\mathsf{int}].X$ and $![\mathsf{int}].\mu X.![\mathsf{int}].X$ to be dual. We therefore define the *coinductive duality relation* $\perp_c$ in a similar way to the coinductive subtyping relation.

Definition 7. *Let SType be the set of all closed type expressions defined by the grammar for S (Figure 1).*

Definition 8. *A relation $R \subseteq SType \times SType$ is a* duality relation *if $(T, U) \in R$ implies the following conditions:*

$$\frac{\Gamma \text{ completed}}{\Gamma \vdash \mathbf{0}}\ \text{T-Nil} \qquad \frac{\Gamma_1 \vdash P \quad \Gamma_2 \vdash Q}{\Gamma_1 + \Gamma_2 \vdash P \mid Q}\ \text{T-Par} \qquad \frac{\Gamma \vdash P \quad \Gamma \text{ unlimited}}{\Gamma \vdash !P}\ \text{T-Rep}$$

$$\frac{\Gamma, x{:}T \vdash P \qquad T \text{ is not a session type}}{\Gamma \vdash (\nu x{:}T)P}\ \text{T-New}$$

$$\frac{\Gamma, x^+{:}S, x^-{:}S' \vdash P \qquad S \perp_c S'}{\Gamma \vdash (\nu x{:}S)P}\ \text{T-NewS}$$

$$\frac{\Gamma, x^p{:}S, \widetilde{y}{:}\widetilde{U} \vdash P \qquad \widetilde{T} \leqslant_c \widetilde{U}}{\Gamma, x^p{:}?[\widetilde{T}].S \vdash x^p?[\widetilde{y}{:}\widetilde{U}].P}\ \text{T-InS} \qquad \frac{\Gamma, x^p{:}S \vdash P \qquad \widetilde{U} \leqslant_c \widetilde{T}}{(\Gamma, x^p{:}![\widetilde{T}].S) + \widetilde{y}^{\widetilde{q}}{:}\widetilde{U} \vdash x^p![\widetilde{y}^{\widetilde{q}}].P}\ \text{T-OutS}$$

$$\frac{\Gamma, x{:}\widehat{\ }[\widetilde{T}], \widetilde{y}{:}\widetilde{U} \vdash P \qquad \widetilde{T} \leqslant_c \widetilde{U}}{\Gamma, x{:}\widehat{\ }[\widetilde{T}] \vdash x?[\widetilde{y}{:}\widetilde{U}].P}\ \text{T-In} \qquad \frac{\Gamma, x{:}\widehat{\ }[\widetilde{T}] \vdash P \qquad \widetilde{U} \leqslant_c \widetilde{T}}{(\Gamma, x{:}\widehat{\ }[\widetilde{T}]) + \widetilde{y}^{\widetilde{q}}{:}\widetilde{U} \vdash x![\widetilde{y}^{\widetilde{q}}].P}\ \text{T-Out}$$

$$\frac{m \leqslant n \quad \forall i \in \{1, \ldots, m\}.(\Gamma, x^p{:}T_i \vdash P_i)}{\Gamma, x^p{:}\&\langle\, l_i : T_i \,\rangle_{1 \leqslant i \leqslant m} \vdash x^p \triangleright \{\, l_i : P_i \,\}_{1 \leqslant i \leqslant n}}\ \text{T-Offer}$$

$$\frac{l = l_i \in \{l_1, \ldots, l_n\} \quad \Gamma, x^p{:}T_i \vdash P}{\Gamma, x^p{:} \oplus \langle\, l_i : T_i \,\rangle_{1 \leqslant i \leqslant n} \vdash x^p \triangleleft l.P}\ \text{T-Choose}$$

Fig. 9 Typing rules

1. *If* $unfold(T) = ?[T_1, \ldots, T_n].S_1$ *then* $unfold(U) = ![U_1, \ldots, U_n].S_2$ *and* $(S_1, S_2) \in R$ *and for all* $i \in \{1, \ldots, n\}$, $T_i \leqslant_c U_i$ *and* $U_i \leqslant_c T_i$.
2. *If* $unfold(T) = ![T_1, \ldots, T_n].S_1$ *then* $unfold(U) = ?[U_1, \ldots, U_n].S_2$ *and* $(S_1, S_2) \in R$ *and for all* $i \in \{1, \ldots, n\}$, $T_i \leqslant_c U_i$ *and* $U_i \leqslant_c T_i$.
3. *If* $unfold(T) = \&\langle\, l_i : S_i \,\rangle_{1 \leqslant i \leqslant n}$ *then* $unfold(U) = \oplus\langle\, l_i : S'_i \,\rangle_{1 \leqslant i \leqslant n}$ *and for all* $i \in \{1, \ldots, n\}$, $(S_i, S'_i) \in R$.
4. *If* $unfold(T) = \oplus\langle\, l_i : S_i \,\rangle_{1 \leqslant i \leqslant n}$ *then* $unfold(U) = \&\langle\, l_i : S'_i \,\rangle_{1 \leqslant i \leqslant n}$ *and for all* $i \in \{1, \ldots, n\}$, $(S_i, S'_i) \in R$.
5. *If* $unfold(T) = \mathsf{end}$ *then* $unfold(U) = \mathsf{end}$.

Definition 9. *The* coinductive duality relation $\perp_c$ *is defined by* $T \perp_c U$ *if and only if there exists a duality relation* R *such that* $(T, U) \in R$.

Definition 10. *Let* Γ *be an environment.*

1. Γ *is* unlimited *if it contains no session types.*
2. Γ *is* completed *if every session type in* Γ *is* end.
3. Γ *is* balanced *if whenever* $x^+ : S \in \Gamma$ *and* $x^- : S' \in \Gamma$ *then* $S \perp_c S'$.

Definition 11. *Addition of a typed name to an environment is defined by*

$$\begin{array}{rl}
\Gamma + x^+ : S = \Gamma, x^+ : S & \text{if } x^+ \notin dom(\Gamma) \text{ and } x \notin dom(\Gamma) \\
 & \text{and } S \text{ is a session type} \\
\Gamma + x^- : S = \Gamma, x^- : S & \text{if } x^- \notin dom(\Gamma) \text{ and } x \notin dom(\Gamma) \\
 & \text{and } S \text{ is a session type} \\
\Gamma + x : T = \Gamma, x : T & \text{if } x \notin dom(\Gamma) \text{ and } x^+ \notin dom(\Gamma) \\
 & \text{and } x^- \notin dom(\Gamma) \\
(\Gamma, x : T) + x : T = \Gamma, x : T & \text{if } T \text{ is not a session type}
\end{array}$$

and is undefined in all other cases. Addition is extended inductively to a partial binary operation on environments.

There are two notable features of the typing rules. The first is the fact that the types of session channels change during a derivation, reflecting the construction of sequences of communication. This can be seen in rules T-InS, T-OutS, T-Offer and T-Choose. As a result, the Type Preservation Theorem (Theorem 1, Section 4) must describe the way in which the types of session channels change during execution of a process.

The second is the way in which the rules guarantee that each end (x^+ or x^-) of a session channel is owned by just one process. This is achieved by means of the addition operation on environments. Addition is a partial operation, and the typing rules which use it have as an implicit hypothesis the requirement that their use of addition is well-defined. In rule T-Par, the construction of $\Gamma_1 + \Gamma_2$ requires that each session channel occurs in at most one of Γ_1 and Γ_2. In rule T-OutS, the construction of $\Gamma + \widetilde{y^q}$ requires that the names $\widetilde{y^q}$ do not occur in Γ and are therefore not used in the continuation process P after they have been sent on x^p. This use of addition is based on the linear type system of Kobayashi *et al.* [18].

Rule T-Nil ensures that a process contains enough communication operations to fully use each session channel. However, it is not possible to guarantee that every session channel is fully used at run-time, because of the possibility of deadlocks. For example, the process

$$x^-?[u{:}\mathsf{int}].y^+![2].\mathbf{0} \mid y^-?[v{:}\mathsf{int}].x^+![3].\mathbf{0}$$

is deadlocked but the typing

$$x^- {:} ?[\mathsf{int}].\mathsf{end}, x^+ {:} ![\mathsf{int}].\mathsf{end}, y^- {:} ?[\mathsf{int}].\mathsf{end}, y^+ {:} ![\mathsf{int}].\mathsf{end} \vdash x^-?[u{:}\mathsf{int}].y^+![2].\mathbf{0} \mid y^-?[v{:}\mathsf{int}].x^+![3].\mathbf{0}$$

indicates that every component of every session type has a matching send or receive operation.

We treat recursive types as definitionally equal to their unfoldings; typing derivations may fold/unfold freely. We incorporate subsumption into the typing rules, rather than by means of a rule such as

$$\frac{\Gamma(x^p) = T \quad T \leqslant_c U}{\Gamma \vdash x^p : U}$$

because it makes the connection between the typing rules and the typechecking algorithm more direct. However, this is not enough in itself to yield a typechecking algorithm: rule T-Par does not specify how to express the environment Γ as $\Gamma_1 + \Gamma_2$ and we have not yet presented an algorithm for checking the subtyping relation between arbitrary types. We address these points in Section 5.

3.5 Example Typing Derivations

Assuming that the necessary data types and typing rules are added, the derivation of the final typing judgement in Section 2.2,

$$a{:}\widehat{\ }[T] \vdash \mathsf{client}$$

where

$$\begin{aligned} \mathsf{clientbody}(x^-) &= x^- \triangleleft \mathsf{plus}.x^-![2].x^-![3].x^-?[u{:}\mathsf{int}].\mathbf{0} \\ \mathsf{client} &= (\nu x : S)(a![x^+].\mathsf{clientbody}(x^-)) \end{aligned}$$

is as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{a{:}\widehat{\ }[T], x^-{:}\mathsf{end} \;\; \text{completed}}{a{:}\widehat{\ }[T], x^-{:}\mathsf{end} \vdash \mathbf{0}}\,\text{T-Nil}}{\vdots}}{\dfrac{a{:}\widehat{\ }[T], x^-{:}![\mathsf{int}].![\mathsf{int}].?[\mathsf{int}].\mathsf{end} \vdash x^-![2]. \ldots \mathbf{0}}{a{:}\widehat{\ }[T], x^-{:} \oplus \langle\, \mathsf{plus}{:}\ldots, \mathsf{eq}{:}\ldots \,\rangle \vdash \mathsf{clientbody}(x^-)}\,\text{T-Choose} \qquad S \leqslant_c T}\,\text{T-Out}}{a{:}\widehat{\ }[T], x^+{:}S, x^-{:}\overline{S} \vdash a![x^+].\mathsf{clientbody}(x^-)}\,\text{T-NewS}}{a{:}\widehat{\ }[T] \vdash (\nu x : S)(a![x^+].\mathsf{clientbody}(x^-))}$$

A key point about this derivation is the change in environment associated with the T-Out rule. To form the process $a![x^+].\mathsf{clientbody}(x^-)$ the typed name $x^+ : S$ must be added to the environment. We have

$$(a{:}\widehat{\ }[T], x^- : \overline{S}) + x^+ : S = a{:}\widehat{\ }[T], x^+ : S, x^- : \overline{S}$$

(the order of names within the environment is not significant).

It is the definition of $+$ which controls whether or not a channel can be used after it has been sent. For example, if $a:\hat{}[![\mathsf{int}].\mathsf{end}]$ and $x:![\mathsf{int}].\mathsf{end}$ then the process $a![x].x![2].\mathbf{0}$ cannot be typed. We have

$$a:\hat{}[![\mathsf{int}].\mathsf{end}], x:\mathsf{end} \vdash \mathbf{0}$$

and using rule T-OutS we can derive

$$a:\hat{}[![\mathsf{int}].\mathsf{end}], x:![\mathsf{int}].\mathsf{end} \vdash x![2].\mathbf{0}.$$

To derive a typing for $a![x].x![2].\mathbf{0}$ we would have to use rule T-Out and this would require adding $x:![\mathsf{int}].\mathsf{end}$ to the environment; but

$$(a:\hat{}[![\mathsf{int}].\mathsf{end}], x:![\mathsf{int}].\mathsf{end}) + x:![\mathsf{int}].\mathsf{end}$$

is not defined.

The situation is different if x is not a session channel. If $a:\hat{}[\hat{}[\mathsf{int}]]$ and $x:\hat{}[\mathsf{int}]$ then we have the derivation

$$\dfrac{\dfrac{a:\hat{}[\hat{}[\mathsf{int}]] \vdash \mathbf{0}}{a:\hat{}[\hat{}[\mathsf{int}]], x:\hat{}[\mathsf{int}] \vdash x![2].\mathbf{0}}\ \text{T-Out}}{a:\hat{}[\hat{}[\mathsf{int}]], x:\hat{}[\mathsf{int}] \vdash a![x].x![2].\mathbf{0}}\ \text{T-Out}$$

which relies on the fact that

$$(a:\hat{}[\hat{}[\mathsf{int}]], x:\hat{}[\mathsf{int}]) + x:\hat{}[\mathsf{int}] = a:\hat{}[\hat{}[\mathsf{int}]], x:\hat{}[\mathsf{int}].$$

3.6 Embedding the Simply Typed Pi Calculus

To demonstrate that our language and type system are an extension of a more standard pi calculus, we briefly show that a simpler language without session types can be embedded in our language.

The following grammar defines non-recursive channel types, and the syntax of a standard pi calculus with synchronous output.

$$\begin{aligned} T &::= \hat{}[T_1, \ldots, T_n] \\ P, Q &::= \mathbf{0} \mid (P \mid Q) \mid !P \mid x?[y_1:T_1, \ldots, y_n:T_n].P \\ &\quad \mid x![y_1, \ldots, y_n].P \mid (\nu x:T)P \end{aligned}$$

The typing rules in Figure 10 define judgements $\Gamma \triangleright P$ in which Γ is a function from names (there are no polarities) to channel types. This is essentially the simple type system for the pi calculus [30, Chapter 6].

Proposition 1. *If $\Gamma \triangleright P$ then $\Gamma \vdash P$.*

$$\frac{}{\Gamma \triangleright \mathbf{0}}\ \text{T-Nil}' \qquad \frac{\Gamma \triangleright P \quad \Gamma \triangleright Q}{\Gamma \triangleright P \mid Q}\ \text{T-Par}'$$

$$\frac{\Gamma \triangleright P}{\Gamma \triangleright\, !P}\ \text{T-Rep}' \qquad \frac{\Gamma, x{:}T \triangleright P}{\Gamma \triangleright (\nu x{:}T)P}\ \text{T-New}'$$

$$\frac{\Gamma, x{:}\widehat{\ }[\widetilde{T}], \widetilde{y}{:}\widetilde{T} \triangleright P}{\Gamma, x{:}\widehat{\ }[\widetilde{T}] \triangleright x?[\widetilde{y}{:}\widetilde{T}].P}\ \text{T-In}' \qquad \frac{\Gamma, x{:}\widehat{\ }[\widetilde{T}], \widetilde{y}{:}\widetilde{T} \triangleright P}{\Gamma, x{:}\widehat{\ }[\widetilde{T}], \widetilde{y}{:}\widetilde{T} \triangleright x![\widetilde{y}].P}\ \text{T-Out}'$$

Fig. 10 Typing rules for the simply typed pi calculus

Proof. By induction on the derivation of $\Gamma \triangleright P$, considering the possibilities for the last rule.

T-Nil′: Γ is completed because it contains no session types, so T-Nil gives $\Gamma \vdash \mathbf{0}$.

T-Par′: By the induction hypothesis, $\Gamma \vdash P$ and $\Gamma \vdash Q$. Because Γ contains no session types we have $\Gamma + \Gamma = \Gamma$. So T-Par gives $\Gamma \vdash P \mid Q$.

T-Rep′: By the induction hypothesis, $\Gamma \vdash P$. Because Γ contains no session types, it is unlimited. So T-Rep gives $\Gamma \vdash !P$.

T-New′: By the induction hypothesis, $\Gamma, x : T \vdash P$. Because T is not a session type, T-New gives $\Gamma \vdash (\nu x{:}T)P$.

T-In′: By the induction hypothesis, $\Gamma, x{:}\widehat{\ }[\widetilde{T}], \widetilde{y}{:}\widetilde{T} \vdash P$. Rule T-In, using reflexivity of $\leqslant_c$ (Proposition 2, Section 4), gives $\Gamma, x : \widehat{\ }[\widetilde{T}] \vdash x?[\widetilde{y}{:}\widetilde{T}].P$.

T-Out′: By the induction hypothesis, $\Gamma, x{:}\widehat{\ }[\widetilde{T}], \widetilde{y}{:}\widetilde{T} \vdash P$. Because there are no session types, $(\Gamma, x{:}\widehat{\ }[\widetilde{T}], \widetilde{y}{:}\widetilde{T}) + \widetilde{y}{:}\widetilde{T} = \Gamma, x{:}\widehat{\ }[\widetilde{T}], \widetilde{y}{:}\widetilde{T}$. So T-Out, using reflexivity of $\leqslant_c$, gives $\Gamma, x{:}\widehat{\ }[\widetilde{T}], \widetilde{y}{:}\widetilde{T} \vdash x![\widetilde{y}].P$. □

4 Soundness of the Type System

The proof that correctly-typed processes have no communication errors at run-time follows a pattern familiar from other type systems for the pi calculus. We first prove some basic properties of the coinductive subtyping and duality relations, then prove a series of results leading to a Type Preservation theorem, and finally state a Type Safety theorem which explicitly shows that a typable process cannot immediately generate a run-time error.

Proposition 2. *The relation $\leqslant_c$ is reflexive.*

Proof. For $T \in \mathit{Type}$ we need to construct a type simulation R such that $(T,T) \in R$. Let $R = \{(U,U) \mid U \in \mathit{Type}\}$. Clearly $(T,T) \in R$, and it is easy to check that R is a type simulation. Consider $(U,V) \in R$. A typical case is that $\mathit{unfold}(U) = ?[U_1,\ldots,U_n].S$. Because $V = U$, we trivially have $\mathit{unfold}(V) = ?[U_1,\ldots,U_n].S$. The definition of R means that $(S,S) \in R$ and for all $i \in \{1,\ldots,n\}$, $(U_i,U_i) \in R$. Therefore case (2) of Definition 3 is satisfied. The other cases are similar. □

Proposition 3. *The relation $\leqslant_c$ is transitive.*

Proof. Suppose that $T \leqslant_c T'$ and $T' \leqslant_c T''$. To prove that $T \leqslant_c T''$ we need to construct a type simulation R such that $(T,T'') \in R$. Let R_1 and R_2 be type simulations such that $(T,T') \in R_1$ and $(T',T'') \in R_2$. Denoting relational composition by $R_1 \cdot R_2$, let

$$\begin{aligned} R &= (R_1 \cdot R_2) \cup (R_2 \cdot R_1) \\ &= \{(T,V) \mid \text{for some } U, (T,U) \in R_1 \text{ and } (U,V) \in R_2\} \\ &\cup \{(V,T) \mid \text{for some } U, (V,U) \in R_2 \text{ and } (U,T) \in R_1\}. \end{aligned}$$

Clearly $(T,T'') \in R$. We need to show that R is a type simulation. Consider $(U,V) \in R$. There are 12 cases: two for each case of Definition 3, depending on whether $(U,V) \in R_1 \cdot R_2$ or $(U,V) \in R_2 \cdot R_1$.

A typical case is $(U,V) \in R_1 \cdot R_2$ and $\mathit{unfold}(U) = ![T_1,\ldots,T_n].S_1$. Then there exists W such that $(U,W) \in R_1$ and $(W,V) \in R_2$. Because R_1 is a type simulation, $\mathit{unfold}(W) = ![T'_1,\ldots,T'_n].S_2$ and $(S_1,S_2) \in R_1$ and for all $i \in \{1,\ldots,n\}$, $(T'_i,T_i) \in R_1$. Because R_2 is a type simulation, $\mathit{unfold}(V) = ![T''_1,\ldots,T''_n].S_3$ and $(S_2,S_3) \in R_2$ and for all $i \in \{1,\ldots,n\}$, $(T''_i,T'_i) \in R_2$. Therefore $(S_1,S_3) \in R_1 \cdot R_2 \subseteq R$ and for all $i \in \{1,\ldots,n\}$, $(T''_i,T_i) \in R_2 \cdot R_1 \subseteq R$. So case (3) of Definition 3 is satisfied.

The other cases are similar. □

Lemma 1. *$\mu X.T \leqslant_c T\{\mu X.T/X\}$ and $T\{\mu X.T/X\} \leqslant_c \mu X.T$.*

Proof. Follows directly from the fact that $\mathit{unfold}(\mu X.T) = \mathit{unfold}(T\{\mu X.T/X\})$. □

Lemma 2 (Inversion). *Suppose that $T \leqslant_c T'$.*

1. *If $\mathit{unfold}(T') = \mathsf{end}$ then $\mathit{unfold}(T) = \mathsf{end}$.*
2. *If $\mathit{unfold}(T') = ?[\widetilde{T}].U$ then $\mathit{unfold}(T) = ?[\widetilde{V}].W$ with $\widetilde{V} \leqslant_c \widetilde{T}$ and $W \leqslant_c U$.*
3. *If $\mathit{unfold}(T') = ![\widetilde{T}].U$ then $\mathit{unfold}(T) = ![\widetilde{V}].W$ with $\widetilde{T} \leqslant_c \widetilde{V}$ and $W \leqslant_c U$.*

4. *If* $unfold(T') = \&\langle\, l_i : T_i \,\rangle_{1 \leqslant i \leqslant n}$ *then* $unfold(T) = \&\langle\, l_i : U_i \,\rangle_{1 \leqslant i \leqslant m}$ *with* $m \leqslant n$ *and* $\forall i \in \{1, \ldots, m\}.U_i \leqslant_c T_i$.
5. *If* $unfold(T') = \oplus\langle\, l_i : T_i \,\rangle_{1 \leqslant i \leqslant n}$ *then* $unfold(T) = \oplus\langle\, l_i : U_i \,\rangle_{1 \leqslant i \leqslant m}$ *with* $n \leqslant m$ *and* $\forall i \in \{1, \ldots, n\}.U_i \leqslant_c T_i$.
6. *If* $unfold(T') = \widehat{\ }[\widetilde{U}]$ *then* $unfold(T) = \widehat{\ }[\widetilde{V}]$ *with* $\widetilde{U} \leqslant_c \widetilde{V}$ *and* $\widetilde{V} \leqslant_c \widetilde{U}$.

Proof. Case (4) is typical; the others are similar. Let R be a type simulation such that $(T, T') \in R$ and suppose that $unfold(T') = \&\langle\, l_i : T_i \,\rangle_{1 \leqslant i \leqslant n} \in R$. In order not to contradict Definition 3, we must have $unfold(T) = \&\langle\, l_i : U_i \,\rangle_{1 \leqslant i \leqslant m}$ with $m \leqslant n$ and $\forall i \in \{1, \ldots, m\}.(U_i, T_i) \in R$. Hence $\forall i \in \{1, \ldots, m\}.U_i \leqslant_c T_i$. □

Lemma 3. *For all* $S_1, S_2 \in SType$, *if* $S_1 \leqslant_c S_2$ *and* $unfold(S_2) \neq \mathsf{end}$ *then it is not the case that* $\overline{S_1} \leqslant_c S_2$.

Proof. Follows from Lemma 2, because only one of S_1 and $\overline{S_1}$ can match the structure of S_2. □

Proposition 4. *The relation* $\perp_c$ *is symmetric.*

Proof. Suppose that $T_1 \perp_c T_2$. To show that $T_2 \perp_c T_1$ we must construct a duality relation R such that $(T_2, T_1) \in R$. Let R' be a duality relation such that $(T_1, T_2) \in R'$. Let $R = \{(U, T) \mid (T, U) \in R'\}$. Clearly $(T_2, T_1) \in R$, and we must show that R is a duality relation.

Suppose that $(U, T) \in R$, meaning that $(T, U) \in R'$. We consider the possibilities for U, according to Definition 8. Case (1) is typical and the others are similar. So suppose that $unfold(U) = ?[U_1, \ldots, U_n].S_2$. By considering the cases of Definition 8 as applied to R', we must have $unfold(T) = ![T_1, \ldots, T_n].S_1$, $(S_1, S_2) \in R'$ and for all $i \in \{1, \ldots, n\}$, $T_i \leqslant_c U_i$ and $U_i \leqslant_c T_i$. Therefore $(S_2, S_1) \in R$, and so we have verified the conditions of case (1). □

Proposition 5. *If* $S \in SType$ *then* $S \perp_c \overline{S}$.

Proof. Let $R = \{(S, \overline{S}) \mid S \in SType\}$. We must show that R is a duality relation. We consider the possibilities for S, according to Definition 8. Case (2) is typical and the others are similar.

If $unfold(S) = ![T_1, \ldots, T_n].S_1$ then $unfold(\overline{S}) = ?[T_1, \ldots, T_n].\overline{S_1}$. We have, for all $i \in \{1, \ldots, n\}$, $T_i \leqslant_c T_i$ by Proposition 2; also, $(S_1, \overline{S_1}) \in R$. □

Lemma 4. *If* $S \perp_c S'$ *then* $tail(S, l) \perp_c tail(S', l)$.

Proof. Let R be a duality relation such that $(S, S') \in R$. Each case of Definition 8 immediately implies that $(tail(S, l), tail(S', l)) \in R$. □

Lemma 5. *If $\Gamma, x^p : S \vdash P$ and S is a session type and $x^p \notin fn(P)$ then $S = \mathsf{end}$.*

Proof. A straightforward induction on the derivation of $\Gamma, x^p : S \vdash P$, ultimately depending on the hypothesis that the environment in T-Nil is completed. □

Lemma 6. *If $\Gamma \vdash P$ and T is not a session type then $\Gamma, x{:}T \vdash P$.*

Proof. A straightforward induction on the derivation of $\Gamma \vdash P$, ultimately depending on the fact that adding a non-session type to a completed or unlimited environment produces an environment which is also completed or unlimited. □

Lemma 7. *If $\Gamma \vdash P$ then $fn(P) \subseteq dom(\Gamma)$.*

Proof. A straightforward induction on the derivation of $\Gamma \vdash P$. □

Lemma 8 (Substitution). *If $\Gamma, w : W \vdash P$ and $Z \leqslant_c W$ and $\Gamma + z^r : T$ is defined then $\Gamma + z^r : Z \vdash P\{z^r/w\}$.*

Proof. By induction on the derivation of $\Gamma, w : W \vdash P$, with a case-analysis on the last rule used. We show the case for T-OutS, meaning that $P = x^p![\widetilde{y^q}].P'$; the others are similar but simpler. For notational simplicity, without losing the essence of the argument, we consider the case in which a single name is output. There are several subcases, depending on the position of w and the form of W. In most cases, transitivity of subtyping (Proposition 3) plays a key role. Note that it is not possible to type a process of the form $w![w].P'$ if w is a session channel, because the instance of T-OutS would require an environment $(\Gamma, w : S) + w : T$, but the addition would not be defined.

1. w is the output name. In this case the derivation ends with

$$\frac{\Gamma, x^p : S \vdash P' \qquad W \leqslant_c T}{(\Gamma, x^p : ![T].S) + w : W \vdash x^p![w].P'} \text{ T-OutS}$$

   By Proposition 3 we have $Z \leqslant_c T$. We now consider the form of W and the relationship of w to Γ.

   (a) W is not a session type. Because $Z \leqslant_c W$ is not a session type, z must be unpolarized.
      i. $w \in dom(\Gamma)$. In this case we have

$$\frac{\Gamma', w : W, x^p : S \vdash P' \qquad W \leqslant_c T}{(\Gamma', w : W, x^p : ![T].S) + w : W \vdash x^p![w].P'}$$

By the induction hypothesis we have

$$\Gamma', z:Z, x^p:S \vdash P'\{z/w\}$$

so T-OutS gives

$$\Gamma', z:Z, x^p:\![T].S \vdash x^p![z].P'\{z/w\}$$

which, because $(x^p![w].P')\{z/w\} = x^p![z].P'\{z/w\}$, is the required judgement.

ii. $w \notin dom(\Gamma)$. In this case we have

$$\frac{\Gamma, x^p:S \vdash P' \qquad W \leqslant_c T}{(\Gamma, w:W, x^p:\![T].S) + w:W \vdash x^p![w].P'}$$

By the induction hypothesis we have

$$\Gamma, z:Z, x^p:S \vdash P'$$

so T-OutS gives

$$\Gamma', z:Z, x^p:\![T].S \vdash x^p![z].P'$$

which, because Lemma 7 implies that $P'\{z/w\} = P'$, is the required judgement.

(b) W is a session type, so $w \notin dom(\Gamma)$. We have

$$\frac{\Gamma, x^p:S \vdash P' \qquad W \leqslant_c T}{\Gamma, w:W, x^p:\![T].S \vdash x^p![w].P'}$$

By the induction hypothesis we have

$$(\Gamma, x^p:S) + z^r:Z \vdash P'\{z^r/w\}$$

and by Lemma 7, $P'\{z^r/w\} = P'$. T-OutS gives

$$(\Gamma, x^p:\![T].S) + z^r:Z \vdash x^p![z^r].P'$$

which is the required judgement.

2. w is the channel used for output. In this case $W = \![T].S$. Because $Z \leqslant_c W$, we have $Z = \![T'].S'$ with $S' \leqslant_c S$ and $T \leqslant_c T'$. The derivation ends with

$$\frac{\Gamma, w:S \vdash P' \qquad U \leqslant_c T}{(\Gamma, w:\![T].S) + y^q:U \vdash w![y^q].P'}$$

By Proposition 3 we have $U \leqslant_c T'$. By the induction hypothesis we have

$$\Gamma + z^r:S' \vdash P'\{z^r/w\}$$

so T-OutS gives

$$\Gamma + z^r : ![T'].S' + y^q : U \vdash z^r![y^q].P'\{z^r/w\}$$

as required.

3. w is not involved in the output. In this case the derivation ends with

$$\frac{\Gamma, w : W, x^p : S \vdash P' \qquad U \leqslant_c T}{(\Gamma, w : W, x^p : ![T].S) + y^q : U \vdash x^p![y^q].P'}$$

The induction hypothesis gives

$$(\Gamma, x^p : S) + z^r : Z \vdash P'\{z^r/w\}$$

and T-OutS gives

$$(\Gamma, x^p : ![T].S) + z^r : Z + y^q : U \vdash x^p![y^q].P'\{z^r/w\}$$

as required. □

Lemma 9 (Structural Congruence Preserves Typing). *If $\Gamma \vdash P$ and $P \equiv Q$ then $\Gamma \vdash Q$.*

Proof. By induction on the derivation of $P \equiv Q$, with a case-analysis on the last rule used. The inductive cases are the congruence rules, and are straightforward. Of the other cases, we show S-Extr, in both directions, in the case involving a session type.

(Left-to-right): We have

$$\dfrac{\dfrac{\Gamma_1, x^+ : S, x^- : S' \vdash P \qquad S \perp_c S'}{\Gamma_1 \vdash (\nu x : S)P}\,\text{T-NewS} \qquad \Gamma_2 \vdash Q}{\Gamma_1 + \Gamma_2 \vdash (\nu x : S)P \mid Q}\,\text{T-Par}$$

which can be rearranged to give

$$\dfrac{\dfrac{\Gamma_1, x^+ : S, x^- : S' \vdash P \qquad \Gamma_2 \vdash Q}{(\Gamma_1 + \Gamma_2), x^+ : S, x^- : S' \vdash P \mid Q \qquad S \perp_c S'}\,\text{T-Par}}{\Gamma_1 + \Gamma_2 \vdash (\nu x : S)(P \mid Q)}\,\text{T-NewS}$$

because we can assume that $x^+, x^- \notin dom(\Gamma_2)$ by the variable convention.

(Right-to-left): We have

$$\dfrac{\dfrac{\Gamma_1 \vdash P \qquad \Gamma_2 \vdash Q}{\Gamma, x^+ : S, x^- : S' \vdash P \mid Q \qquad S \perp_c S'}\,\text{T-Par}}{\Gamma \vdash (\nu x : S)(P \mid Q)}\,\text{T-NewS}$$

where $\Gamma_1 + \Gamma_2 = \Gamma, x^+ : S, x^- : S'$. Because $x^+ \notin fn(Q)$ and $x^- \notin fn(Q)$, and $S \neq \mathsf{end}$ (hence also $S' \neq \mathsf{end}$), Lemma 5 gives $x^+, x^- \notin dom(\Gamma_2)$. Therefore $\Gamma_1 = \Gamma_1', x^+ : S, x^- : S'$ with $\Gamma_1' + \Gamma_2 = \Gamma$. We can construct the derivation

$$\dfrac{\dfrac{\Gamma_1', x^+ : S, x^- : S' \vdash P \qquad S \perp_c S'}{\Gamma_1' \vdash (\nu x : S)P}\ \text{T-NewS} \qquad \Gamma_2 \vdash Q}{\Gamma \vdash (\nu x : S)P \mid Q}\ \text{T-Par}$$

□

Theorem 1 (Type Preservation).

1. *If $\Gamma \vdash P$ and $P \xrightarrow{\tau,-} Q$ then $\Gamma \vdash Q$.*
2. *If $\Gamma, x^+ : S, x^- : S' \vdash P$ and $S \perp_c S'$ and $P \xrightarrow{x,l} Q$ then $\Gamma, x^+ : tail(S, l), x^- : tail(\overline{S}, l) \vdash Q$.*
3. *If $\Gamma, x : T \vdash P$ and $P \xrightarrow{x,-} Q$ then $\Gamma, x : T \vdash Q$.*

Proof. By induction on the derivation of the reduction, considering the cases appropriate for the form of the label.

1. The important case is when the derivation of the reduction ends with R-NewS. We have

$$\dfrac{P \xrightarrow{x,l} P'}{(\nu x : S)P \xrightarrow{\tau,-} (\nu x : tail(S, l))P'}\ \text{R-NewS}$$

and the derivation of $\Gamma \vdash (\nu x : S)P$ ends with

$$\dfrac{\Gamma, x^+ : S, x^- : S' \vdash P \qquad S \perp_c S'}{\Gamma \vdash (\nu x : S)P}\ \text{T-NewS}$$

By the induction hypothesis (clause 2) we have

$$\Gamma, x^+ : tail(S, l), x^- : tail(S', l) \vdash P'$$

By Lemma 4, $tail(S, l) \perp_c tail(S', l)$. So T-NewS gives

$$\Gamma \vdash (\nu x : tail(S, l))P'$$

as required.
The cases for R-New, R-Par and R-Cong follow straightforwardly from the induction hypothesis, using Lemma 9 for R-Cong.

2. There are two important cases: R-Com and R-Select. For R-Com we have
$$x^p?[\widetilde{y}{:}\widetilde{T}].P \mid x^{\bar{p}}![\widetilde{z}^{\widetilde{q}}].Q \xrightarrow{x,-} P\{\widetilde{z}^{\widetilde{q}}/\widetilde{y}\} \mid Q$$
and the form of the environment $\Gamma, x^+{:}S, x^-{:}S'$ means that p is either $^+$ or $^-$; without loss of generality assume $p = {}^+$. The derivation of $\Gamma, x^+{:}S, x^-{:}S' \vdash x^+?[\widetilde{y}{:}\widetilde{T}].P \mid x^-![\widetilde{z}^{\widetilde{q}}].Q$ ends as follows; note that S must be of the form $?[\widetilde{U}].S_1$ with $\widetilde{U} \leqslant_c \widetilde{T}$ and S' must be of the form $![\widetilde{U}].S_2$ with $S_1 \perp_c S_2$, and $\Gamma_1 + \Gamma_2 = \Gamma$, and $\Gamma_3 + \widetilde{z}^{\widetilde{q}}{:}\widetilde{V} = \Gamma_2$, and $\widetilde{V} \leqslant_c \widetilde{U}$.
$$\dfrac{\dfrac{\Gamma_1, x^+{:}S_1, \widetilde{y}{:}\widetilde{T} \vdash P}{\Gamma_1, x^+{:}?[\widetilde{U}].S_1 \vdash x^+?[\widetilde{y}{:}\widetilde{T}].P} \qquad \dfrac{\Gamma_3, x^-{:}S_2 \vdash Q}{\Gamma_2, x^-{:}![\widetilde{U}].S_2 \vdash x^-![\widetilde{z}^{\widetilde{q}}].Q}}{\Gamma, x^+{:}?[\widetilde{U}].S_1, x^-{:}![\widetilde{U}].S_2 \vdash x^+?[\widetilde{y}{:}\widetilde{T}].P \mid x^-![\widetilde{z}^{\widetilde{q}}].Q}$$
By transitivity (Proposition 3) we have $\widetilde{V} \leqslant_c \widetilde{T}$, and Lemma 8 gives
$$(\Gamma_1, x^+{:}S_1) + \widetilde{z}^{\widetilde{q}}{:}\widetilde{V} \vdash P\{\widetilde{z}^{\widetilde{q}}/y\}.$$
By using T-Par we can derive
$$(\Gamma_1, x^+{:}S_1) + \widetilde{z}^{\widetilde{q}}{:}\widetilde{V} + (\Gamma_3, x^-{:}S_2) \vdash P\{\widetilde{z}^{\widetilde{q}}/y\} \mid Q$$
which is the desired judgement, because $\Gamma_1 + \widetilde{z}^{\widetilde{q}}{:}\widetilde{V} + \Gamma_3 = \Gamma$. The case of R-Select is similar, but simpler because there is no substitution. The cases of R-Par, R-New and R-Cong again follow straightforwardly from the induction hypothesis.
3. The main case is R-Com, which follows by similar but simpler reasoning to that for clause (2). The other cases follow straightforwardly from the induction hypothesis. □

We now prove that a correctly-typed process contains no immediate possibilities for a communication error. We need to assume that the process is typed in a balanced environment; note that the Type Preservation theorem guarantees that the property of being typable in a balanced environment is preserved by reductions.

Theorem 2 (Type Safety). *Let $\Gamma \vdash P$ where Γ is balanced.*

1. *If $P \equiv (\nu\widetilde{u}{:}\widetilde{T})(x?[\widetilde{y}{:}\widetilde{V}].P_1 \mid x![\widetilde{z}^{\widetilde{q}}].P_2 \mid Q)$ then among $\Gamma, \widetilde{u}{:}\widetilde{T}$ we have $x{:}\widehat{\ }[\widetilde{U}]$ and $\widetilde{z}^{\widetilde{q}}{:}\widetilde{W}$ with $\widetilde{W} \leqslant \widetilde{U} \leqslant \widetilde{V}$.*
2. *If $P \equiv (\nu\widetilde{u}{:}\widetilde{T})(x^p?[\widetilde{y}{:}\widetilde{V}].P_1 \mid x^{\bar{p}}![\widetilde{z}^{\widetilde{q}}].P_2 \mid Q)$ with $p = {}^+$ or $p = {}^-$ then $x^+, x^- \notin fn(Q)$ and among $\Gamma, \widetilde{u}{:}\widetilde{T}$ we have $x^p{:}?[\widetilde{U}].S$ and $x^{\bar{p}}{:}![\widetilde{U}].\overline{S}$ and $\widetilde{z}^{\widetilde{q}}{:}\widetilde{W}$ with $\widetilde{W} \leqslant \widetilde{U} \leqslant \widetilde{V}$.*

3. If $P \equiv (\nu \widetilde{u}{:}\widetilde{T})(x^{p} \triangleright \{\ l_i : P_i\ \}_{1 \leqslant i \leqslant n} \mid x^{\overline{p}} \triangleleft l.Q \mid R)$ *then* $l \in \{l_1, \ldots, l_n\}$ *and* $x^+, x^- \notin \mathit{fn}(R)$.

Proof. By analyzing the final steps in the derivation of $\Gamma \vdash P$, and using the information in the hypotheses of the typing rules used. □

At the top level, a process P is typechecked in a balanced environment; in practice we would expect a complete program to be typechecked in the empty environment, or perhaps an environment containing globally-defined standard channels but no session channels. The Type Preservation Theorem guarantees that as P reduces, each subsequent process Q is typable in a balanced environment, and the Type Safety Theorem guarantees that Q has no immediate communication errors.

5 Typechecking and Inferring Polarities

We now convert the typing rules of Figure 9 into a typechecking algorithm. There are three significant issues to address.

1. We need an algorithm for checking the subtyping relation $\leqslant_c$ between arbitrary types.
2. Rule T-Par does not specify how to express the environment Γ as $\Gamma_1 + \Gamma_2$, so reading it upwards does not directly form part of a syntax-directed typechecking algorithm.
3. We want to eliminate polarities from the syntax, and infer them during typechecking.

Before dealing with these points, we briefly mention two minor issues. The first is that although the typing rules are not completely syntax-directed, distinctions between T-In and T-InS, T-Out and T-OutS, and T-New and T-NewS can easily be made on the basis of type information in the environment or the ν-binding. The second is that we do not need an algorithm for checking the duality relation $\perp_c$ between arbitrary types; when incorporating rule T-NewS into the typechecking algorithm, we replace S' by $\overline{S}$.

5.1 Algorithmic Subtyping

To obtain an algorithm for checking the subtyping relation, we follow the approach taken by Pierce and Sangiorgi [25] in their system of input/output subtyping for the pi calculus. Algorithmic issues in subtyping recursive types are discussed at greater length by Pierce

$$\frac{T \leqslant U \in \Sigma}{\Sigma \vdash T \leqslant U}\ \text{AS-Assump} \qquad \frac{}{\Sigma \vdash \mathsf{end} \leqslant \mathsf{end}}\ \text{AS-End}$$

$$\frac{\Sigma, \mu X.T \leqslant U \vdash T\{\mu X.T/X\} \leqslant U}{\Sigma \vdash \mu X.T \leqslant U}\ \text{AS-RecL}$$

$$\frac{\Sigma, T \leqslant \mu X.U \vdash T \leqslant U\{\mu X.U/X\}}{\Sigma \vdash T \leqslant \mu X.U}\ \text{AS-RecR}$$

$$\frac{\Sigma \vdash \widetilde{T} \leqslant \widetilde{U} \quad \Sigma \vdash V \leqslant W}{\Sigma \vdash ?[\widetilde{T}].V \leqslant ?[\widetilde{U}].W}\ \text{AS-InS} \qquad \frac{\Sigma \vdash \widetilde{U} \leqslant \widetilde{T} \quad \Sigma \vdash V \leqslant W}{\Sigma \vdash ![\widetilde{T}].V \leqslant ![\widetilde{U}].W}\ \text{AS-OutS}$$

$$\frac{m \leqslant n \quad \forall i \in \{1, \ldots, m\}.(\Sigma \vdash S_i \leqslant T_i)}{\Sigma \vdash \&\langle\, l_i : S_i \,\rangle_{1 \leqslant i \leqslant m} \leqslant \&\langle\, l_i : T_i \,\rangle_{1 \leqslant i \leqslant n}}\ \text{AS-Branch}$$

$$\frac{m \leqslant n \quad \forall i \in \{1, \ldots, m\}.(\Sigma \vdash S_i \leqslant T_i)}{\Sigma \vdash \oplus\langle\, l_i : S_i \,\rangle_{1 \leqslant i \leqslant n} \leqslant \oplus\langle\, l_i : T_i \,\rangle_{1 \leqslant i \leqslant m}}\ \text{AS-Choice}$$

Fig. 11 Algorithmic subtyping rules

[26, Chapter 21]. The inference rules in Figure 11 define judgements $\Sigma \vdash T \leqslant U$, in which T and U are types and Σ is a sequence of assumed instances of the subtyping relation. The rules AS-RecL and AS-RecR add assumptions to Σ in order to limit the unfolding of recursive types. In rules AS-InS and AS-OutS we write $\Sigma \vdash \widetilde{T} \leqslant \widetilde{U}$ for $\forall i.(\Sigma \vdash T_i \leqslant U_i)$. If $\emptyset \vdash T \leqslant U$ is derivable then we write $\vdash T \leqslant U$ or just $T \leqslant U$.

We obtain an algorithm for checking the *algorithmic subtyping relation* $\leqslant$ by reading these inference rules upwards, with two additional specifications. First, in order to guarantee termination, AS-Assump should always be used if it is applicable. Second, in order to make the algorithm deterministic, we arbitrarily specify that AS-RecL should be used in preference to AS-RecR if they are both applicable. At the top level, the algorithm is applied to the initial goal $\emptyset \vdash T \leqslant U$.

We now prove that the subtyping algorithm is sound and complete with respect to the coinductive definition of the relation $\leqslant_c$. The proof is based on that of Pierce and Sangiorgi [25]; the main differences are

due to the fact that we have defined $\leqslant_c$ syntactically, by means of *unfold*, rather than by defining a completely unfolded tree for each type.

Lemma 10. *The subtyping algorithm always terminates.*

Proof. Given a type T, define $Sub(T)$ to be the set of all subterms of T, with free type variables replaced by their (recursive) definitions. $Sub(T)$ is defined recursively on the structure of T, as follows. Note that for any T, $Sub(T)$ is finite because its size is bounded by the number of distinct subterms of T.

$$\begin{aligned}
Sub(\hat{}[T_1, \ldots, T_n]) &= \{\hat{}[T_1, \ldots, T_n]\} \\
&\cup Sub(T_1) \cup \ldots \cup Sub(T_n) \\
Sub(?[T_1, \ldots, T_n].S) &= \{?[T_1, \ldots, T_n].S\} \cup Sub(S) \\
&\cup Sub(T_1) \cup \ldots \cup Sub(T_n) \\
Sub(![T_1, \ldots, T_n].S) &= \{![T_1, \ldots, T_n].S\} \cup Sub(S) \\
&\cup Sub(T_1) \cup \ldots \cup Sub(T_n) \\
Sub(\&\langle\, l_1 : S_1, \ldots, l_n : S_n \,\rangle) &= \{\&\langle\, l_1 : S_1, \ldots, l_n : S_n \,\rangle\} \\
&\cup Sub(S_1) \cup \ldots \cup Sub(S_n) \\
Sub(\oplus\langle\, l_1 : S_1, \ldots, l_n : S_n \,\rangle) &= \{\oplus\langle\, l_1 : S_1, \ldots, l_n : S_n \,\rangle\} \\
&\cup Sub(S_1) \cup \ldots \cup Sub(S_n) \\
Sub(\mu X.T) &= \{\mu X.T\} \\
&\cup \{U\{\mu X.T/X\} \mid U \in Sub(T)\} \\
Sub(\mathsf{end}) &= \{\mathsf{end}\} \\
Sub(X) &= \{X\}
\end{aligned}$$

Consider applying the algorithm to the input $\emptyset \vdash T \leqslant U$, and let $Sub(T, U) = Sub(T) \cup Sub(U)$. Let $\Sigma \vdash T' \leqslant U'$ be a goal arising during execution of the algorithm. We prove that the following properties hold for all such goals.

1. $T' \in Sub(T, U)$ and $U' \in Sub(T, U)$.
2. For every assumption $V_1 \leqslant V_2 \in \Sigma$, $V_1 \in Sub(T, U)$ and $V_2 \in Sub(T, U)$.
3. Σ contains no repeated assumptions: if $V_1 \leqslant V_2 \in \Sigma$ and $V_1' \leqslant V_2' \in \Sigma$ then either $V_1 \neq V_1'$ or $V_2 \neq V_2'$.

These properties clearly hold for the initial call $\emptyset \vdash T \leqslant U$, and it is straightforward to check that if the current call satisfies them then so do the new subgoals.

We now define a measure $\mathcal{M}(\Sigma \vdash T \leqslant U)$ on recursive calls, by

$$\mathcal{M}(\Sigma \vdash T \leqslant U) = (n, m)$$

where n is the number of assumptions in Σ and m is the maximum nesting of type constructors in either T or U. Measures are ordered by

$$(n, m) > (n', m') \text{ if } n < n' \text{ or } (n = n' \text{ and } m > m').$$

It is straightforward to check that the application of a rule by the algorithm generates subgoals with smaller measures than the current goal. The ordering of measures is well-founded, because the first component is bounded above by the cardinality of $Sub(T, U) \times Sub(T, U)$, which is finite, and the second component is bounded below by 1. Therefore the algorithm always terminates. □

Definition 12. *A goal $\Sigma \vdash T \leqslant U$ is* sound *if $T \leqslant_c U$ and for all $V_1 \leqslant V_2 \in \Sigma$, $V_1 \leqslant_c V_2$.*

Lemma 11. *If a goal is sound then the conclusion of one of the rules in Figure 11 matches the goal and the new subgoals corresponding to the hypotheses of the rule are all sound goals.*

Proof. Let $\Sigma \vdash T \leqslant U$ be a sound goal. If $T \leqslant U \in \Sigma$ then rule AS-Assump applies and there are no subgoals. If either AS-RecL or AS-RecR applies then soundness of the new subgoal follows from soundness of the original goal and Lemma 1. In the other cases, soundness of the new subgoals follows from the definition of $\leqslant_c$. □

Lemma 12. *If $T \leqslant_c U$ then the subtyping algorithm does not return* false *when applied to $\Sigma \vdash T \leqslant U$.*

Proof. Consider the derivation produced by the algorithm when given $\emptyset \vdash T \leqslant U$ as input, assuming that $T \leqslant_c U$. The initial goal is sound; by Lemma 11 all of the generated subgoals are sound; and again by Lemma 11, when given a sound goal the algorithm can always either proceed or return *true*. □

Theorem 3. *If $T \leqslant_c U$ then $\vdash T \leqslant U$.*

Proof. By Lemma 10 the algorithm terminates. By Lemma 12 the algorithm does not return *false*. Therefore it must return *true*. □

Lemma 13. *If $\Sigma \vdash T \leqslant U$ then $\Sigma, \Sigma' \vdash T \leqslant U$ for any Σ'.*

Proof. By induction on the derivation of $\Sigma \vdash T \leqslant U$. □

Lemma 14. *1. Suppose* $\vdash \mu X.T \leqslant U$ *and* $\mu X.T \leqslant U \notin \Sigma$. *Then* $\Sigma, \mu X.T \leqslant U \vdash V_1 \leqslant V_2$ *implies* $\Sigma \vdash V_1 \leqslant V_2$.
2. Suppose $\vdash T \leqslant \mu X.U$ *and* $T \leqslant \mu X.U \notin \Sigma$. *Then* $\Sigma, T \leqslant \mu X.U \vdash V_1 \leqslant V_2$ *implies* $\Sigma \vdash V_1 \leqslant V_2$.

Proof. We prove (1) by induction on the derivation of $\Sigma, \mu X.T \leqslant U \vdash V_1 \leqslant V_2$. The proof of (2) is similar.

If the last rule used in the derivation is AS-Assump then there are two cases.

1. $V_1 \leqslant V_2 \in \Sigma$. In this case, AS-Assump also gives $\Sigma \vdash V_1 \leqslant V_2$.
2. $V_1 = \mu X.T$ and $V_2 = U$. By hypothesis $\vdash \mu X.T \leqslant U$, hence by Lemma 13 we have $\Sigma \vdash \mu X.T \leqslant U$.

If the last rule in the derivation is not AS-Assump, then the induction hypothesis allows $T \leqslant \mu X.U$ to be removed from the assumptions of the hypotheses, and the same rule then gives a derivation of $\Sigma \vdash V_1 \leqslant V_2$. □

Lemma 15. *1. If* $\vdash \mu X.T \leqslant U$ *then* $\vdash T\{\mu X.T/X\} \leqslant U$.
2. If $\vdash T \leqslant \mu X.U$ *then* $\vdash T \leqslant U\{\mu X.U/X\}$.

Proof. We prove (1); the proof of (2) is similar. The last rule used in the derivation of $\vdash \mu X.T \leqslant U$ is AS-RecL with hypothesis $\mu X.T \leqslant U \vdash T\{\mu X.T/X\} \leqslant U$. By applying Lemma 14 to this hypothesis we have $\vdash T\{\mu X.T/X\} \leqslant U$. □

Corollary 1. *If* $\vdash T \leqslant U$ *then* $\vdash \mathit{unfold}(T) \leqslant \mathit{unfold}(U)$.

Theorem 4. *If* $\vdash T \leqslant U$ *then* $T \leqslant_c U$.

Proof. By Corollary 1 and the fact that $\leqslant_c$ is defined in terms of the unfolded structure of types, it is sufficient to consider the case in which T and U are guarded types. We show that

$$R = \{(T, U) \mid \ \vdash T \leqslant U \text{ and } T \text{ and } U \text{ are guarded}\}$$

is a type simulation. We give the details of case (4) of Definition 3; the other cases are similar.

Assume that $(T, U) \in R$ and $\mathit{unfold}(T) = \&\langle l_1 : S_1, \ldots, l_m : S_m \rangle$. Because T is guarded, this means that $T = \&\langle l_1 : S_1, \ldots, l_m : S_m \rangle$. Because U is also guarded, the last rule in the derivation of $\vdash T \leqslant U$ must be AS-Branch, meaning that $U = \&\langle l_1 : S'_1, \ldots, l_n : S'_n \rangle$ with $m \leqslant n$ and $\forall i \in \{1, \ldots, m\}.(\vdash S_i \leqslant S'_i)$. Corollary 1 gives, for each i, $\vdash \mathit{unfold}(S_i) \leqslant \mathit{unfold}(S'_i)$, and so $(S_i, S'_i) \in R$. □

Corollary 2. $\vdash T \leqslant U$ *if and only if* $T \leqslant_c U$.

5.2 Typechecking

We now have an algorithm for checking the subtyping relation. The next issue in typechecking is how to calculate an appropriate representation of Γ as $\Gamma_1 + \Gamma_2$ in order to implement the rule T-Par. This point has arisen in other settings involving linear type systems [18, 19] and we adopt the same solution: the typechecking algorithm must calculate the set of session channel names used by each process, so that when typechecking $P \mid Q$, the session channels used by P are removed from the environment before typechecking Q. We formalize the typechecking algorithm as a collection of inference rules for judgements $\Gamma \vdash_X P{:}Y$, where Γ, X and P are inputs to the typechecking function and Y is calculated. Here X and Y are sets of polarized or unpolarized names. Y is the set of names which are used by P, either for communication or in messages. X is used to record the names which are available for use by the current thread; when x^p is used for the first time, $x^{\overline{p}}$ is removed from X. At the top level, the typechecking function is called with $X = dom(\Gamma)$.

The final aspect of the typechecking algorithm is that by imposing some additional constraints on the type system, we are able to eliminate polarities from the syntax of processes; they can be inferred during typechecking. However, it is not possible to eliminate polarities from our theory altogether: the Type Preservation theorem of Section 4 can only be proved if polarities are included. The explanation is that in order to infer polarities, we must add extra hypotheses to the typing rules. This has the effect of eliminating some typable processes. If the typing rules are modified throughout the system, then Type Preservation no longer holds. However, if we use the modified typing rules at the top level only, then the processes which are no longer typable all contain immediate deadlocks, and we consider it acceptable to eliminate them. Before presenting the typechecking algorithm, we will now explain this point a little further.

The essential idea for eliminating polarities is to add the hypothesis $x^{\overline{p}} \notin \Gamma$ to the rules T-InS, T-OutS, T-Offer and T-Choose. This ensures that every environment in a typing derivation is balanced. It means that when a process owns both x^+ and x^- — for example, immediately inside (νx) — one end of x cannot be used for communication until the other end has been sent as part of a message. This enforces the intuitive principle that each end of a session channel is used by just one parallel thread within a system. It also means that whenever a polarized channel name x^p occurs at the top level of a process, p is uniquely determined by the current environment Γ: if x^p is used for communication then either $x^{\overline{p}} \notin \Gamma$, or x^p and $x^{\overline{p}}$ have

dual types of which only one matches the communication operation in question. If x^p occurs as part of a message, then it is not possible for both x^p and $x^{\overline{p}}$ to have types which match the channel type. Therefore polarities can be inferred.

An example of a process which cannot be typed with the modified rules is

$$x^+![\,].x^-?[\,].\mathbf{0} \tag{1}$$

Given synchronous communication and linear control of session channels, a process which use both ends of a session channel within the same thread is deadlocked, and although our type system does not aim to eliminate deadlocks in general, it seems harmless to exclude some deadlocks.

However, because our type system is not powerful enough to eliminate deadlocks in general, modifying the rules in this way throughout the language would result in a failure of Type Preservation. For example, if $S = ?[\,].\mathsf{end}$ then the process

$$z^+![x^-].z^+?[y{:}S].x^+![\,].y?[\,].\mathbf{0} \mid z^-?[u{:}S].z^-![u].\mathbf{0}$$

is typable in the environment

$$x^+{:}\overline{S}, x^-{:}S, z^+{:}![S].?[S].\mathsf{end}, z^-{:}?[S].![S].\mathsf{end}$$

but reduces in two steps to (1), which cannot be typed without introducing an unbalanced environment during the derivation.

Our typechecking algorithm is defined by the inference rules in Figures 12 and 13. Typability in this system implies typability in the original type system (Theorem 5), so typable processes are guaranteed to execute safely. Note that the transformation from the typing rules of Figure 9 to the algorithm is independent of subtyping and would also apply to a language with session types but no subtyping. However, the details of developing the algorithm and eliminating polarities have not been published before.

Some definitions and results are needed in order to understand the operation of the typechecking algorithm. The inference rules define judgements $\Gamma \vdash_X P{:}Y$. If this judgement is derivable then the algorithm calculates Y from Γ, X and P. Γ is an environment of the same form as in Section 3; in particular, note that names in $dom(\Gamma)$ may have polarities. X is a subset of $dom(\Gamma)$, specifying which session channels are available for use by P. P is a process in which all names are unpolarized; any polarities which are necessary are calculated by the algorithm and recorded by manipulating the environment and the set X. Y, which is returned by the algorithm, is a subset of X which indicates the session channels which are used by P.

$$\frac{}{\Gamma \vdash_X \mathbf{0}{:}\{x^p \in X \mid \Gamma(x^p) = \mathsf{end}\}}\ \text{TC-Nil}$$

$$\frac{\Gamma \vdash_X P{:}Y \quad \Gamma - Y \vdash_{X-Y} Q{:}Z}{\Gamma \vdash_X P \mid Q{:}Y \cup Z}\ \text{TC-Par}$$

$$\frac{\Gamma \vdash_\emptyset P{:}\emptyset}{\Gamma \vdash_X !P{:}\emptyset}\ \text{TC-Rep} \qquad \frac{\Gamma, x{:}T \vdash_X P{:}Y}{\Gamma \vdash_X (\nu x{:}T)P{:}Y}\ \text{TC-New}$$

$$\frac{\Gamma, x^+{:}S, x^-{:}\overline{S} \vdash_{X \cup \{x^+, x^-\}} P{:}Y \quad \{x^+, x^-\} \subseteq Y \quad \mathsf{end} \not\leqslant S \in \mathit{SType}}{\Gamma \vdash_X (\nu x{:}S)P{:}Y - \{x^+, x^-\}}\ \text{TC-NewS}$$

$$\frac{\Gamma, x{:}\widehat{\ }[\widetilde{T}], \widetilde{y}{:}\widetilde{U} \vdash_{X \cup Y^S} P{:}Y \quad Y^S \subseteq Y \quad \widetilde{T} \leqslant \widetilde{U}}{\Gamma, x{:}\widehat{\ }[\widetilde{T}] \vdash_X x?[\widetilde{y}{:}\widetilde{U}].P{:}Y - Y^S}\ \text{TC-In}$$

$$\frac{(\Gamma, x{:}\widehat{\ }[\widetilde{T}]) - \widetilde{y}^{\tilde{q}}{:}\widetilde{U} \vdash_{X-Y^S} P{:}Y \quad \widetilde{U} \leqslant \widetilde{T} \quad Y^S \subseteq X}{\Gamma, x{:}\widehat{\ }[\widetilde{T}] \vdash_X x![\widetilde{y}].P{:}Y \cup \{\widetilde{y}^{\tilde{q}}\}}\ \text{TC-Out}$$

In TC-In, $Y^S = \{y_i \mid U_i \in \mathit{SType}\}$. In TC-Out, $Y^S = \{y_i^{q_i} \mid U_i \in \mathit{SType}\}$.

Fig. 12 Inference rules for the typechecking algorithm

The rules in Figures 12 and 13 are not syntax-directed: there are four possible rules for each of input and output, and three possible rules for each of offer and choose. We will explain below how a choice of rule is uniquely determined in each case by type information in Γ.

Several rules make use of the operations $\Gamma - \Gamma'$, $\Gamma - Y$ where Y is a set of names, and $X - Y$ where X and Y are sets of names. We also need the operation $\Gamma|_X$ where X is a set of names, in order to state some important invariants of the algorithm.

Definition 13. *The partial operation of subtraction on environments and typed names is defined as follows:*

$$\begin{array}{rl} (\Gamma, x{:}T) - x{:}T = \Gamma, x{:}T & \text{if } T \text{ is not a session type} \\ (\Gamma, x^p{:}S) - x^p{:}S = \Gamma & \text{if } S \text{ is a session type} \end{array}$$

and is undefined in all other cases. Subtraction is extended inductively to a partial operation on environments.

Definition 14. *If Γ is an environment and Y is a set of optionally polarized names, then $\Gamma - Y = \{x^p{:}T \in \Gamma \mid x^p \notin Y\}$.*

Definition 15. *If X and Y are sets of optionally polarized names, then $X - Y = \{x^p \in X \mid x^p \notin Y\}$.*

$$\frac{\Gamma, x{:}S, \widetilde{y}{:}\widetilde{U} \vdash_{X \cup Y^S} P{:}Y \quad Y^S \subseteq Y \quad \widetilde{T} \leqslant \widetilde{U} \quad x \in X \cap Y}{\Gamma, x{:}?[\widetilde{T}].S \vdash_X x?[\widetilde{y}{:}\widetilde{U}].P{:}Y - Y^S} \text{TC-InS}_1$$

$$\frac{\Gamma, x^+{:}S, \widetilde{y}{:}\widetilde{U} \vdash_{(X \cup Y^S) - \{x^-\}} P{:}Y \quad Y^S \subseteq Y \quad \widetilde{T} \leqslant \widetilde{U} \quad x^+ \in X \cap Y}{\Gamma, x^+{:}?[\widetilde{T}].S \vdash_X x?[\widetilde{y}{:}\widetilde{U}].P{:}Y - Y^S} \text{TC-InS}_2$$

$$\frac{\Gamma, x^-{:}S, \widetilde{y}{:}\widetilde{U} \vdash_{(X \cup Y^S) - \{x^+\}} P{:}Y \quad Y^S \subseteq Y \quad \widetilde{T} \leqslant \widetilde{U} \quad x^- \in X \cap Y}{\Gamma, x^-{:}?[\widetilde{T}].S \vdash_X x?[\widetilde{y}{:}\widetilde{U}].P{:}Y - Y^S} \text{TC-InS}_3$$

$$\frac{(\Gamma - \widetilde{y^q}{:}\widetilde{U}), x{:}S \vdash_{X - Y^S} P{:}Y \quad \widetilde{U} \leqslant \widetilde{T} \quad x \in X \cap Y \quad Y^S \subseteq X}{\Gamma, x{:}![\widetilde{T}].S \vdash_X x![\widetilde{y}].P{:}Y \cup Y^S} \text{TC-OutS}_1$$

$$\frac{(\Gamma - \widetilde{y^q}{:}\widetilde{U}), x^+{:}S \vdash_{X - \{x^-\} - Y^S} P{:}Y \quad \widetilde{U} \leqslant \widetilde{T} \quad x^+ \in X \cap Y \quad Y^S \subseteq X}{\Gamma, x^+{:}![\widetilde{T}].S \vdash_X x![\widetilde{y}].P{:}Y \cup Y^S} \text{TC-OutS}_2$$

$$\frac{(\Gamma - \widetilde{y^q}{:}\widetilde{U}), x^-{:}S \vdash_{X - \{x^+\} - Y^S} P{:}Y \quad \widetilde{U} \leqslant \widetilde{T} \quad x^- \in X \cap Y \quad Y^S \subseteq X}{\Gamma, x^-{:}![\widetilde{T}].S \vdash_X x![\widetilde{y}].P{:}Y \cup Y^S} \text{TC-OutS}_3$$

$$\frac{m \leqslant n \quad \forall i \in \{1, \ldots, m\}.(\Gamma, x{:}T_i \vdash_X P_i{:}Y) \quad x \in X \cap Y}{\Gamma, x{:}\&\langle\, l_i : T_i \,\rangle_{1 \leqslant i \leqslant m} \vdash_X x \triangleright \{\, l_i : P_i \,\}_{1 \leqslant i \leqslant n}{:}Y} \text{TC-Offer}_1$$

$$\frac{m \leqslant n \quad \forall i \in \{1, \ldots, m\}.(\Gamma, x^+{:}T_i \vdash_{X - \{x^-\}} P_i{:}Y) \quad x^+ \in X \cap Y}{\Gamma, x^+{:}\&\langle\, l_i : T_i \,\rangle_{1 \leqslant i \leqslant m} \vdash_X x \triangleright \{\, l_i : P_i \,\}_{1 \leqslant i \leqslant n}{:}Y} \text{TC-Offer}_2$$

$$\frac{m \leqslant n \quad \forall i \in \{1, \ldots, m\}.(\Gamma, x^-{:}T_i \vdash_{X - \{x^+\}} P_i{:}Y) \quad x^- \in X \cap Y}{\Gamma, x^-{:}\&\langle\, l_i : T_i \,\rangle_{1 \leqslant i \leqslant m} \vdash_X x \triangleright \{\, l_i : P_i \,\}_{1 \leqslant i \leqslant n}{:}Y} \text{TC-Offer}_3$$

$$\frac{\Gamma, x{:}T_i \vdash_X P{:}Y \quad l = l_i \in \{l_1, \ldots, l_n\} \quad x \in X \cap Y}{\Gamma, x{:} \oplus \langle\, l_i : T_i \,\rangle_{1 \leqslant i \leqslant n} \vdash_X x \triangleleft l.P{:}Y} \text{TC-Choose}_1$$

$$\frac{\Gamma, x^+{:}T_i \vdash_{X - \{x^-\}} P{:}Y \quad l = l_i \in \{l_1, \ldots, l_n\} \quad x^+ \in X \cap Y}{\Gamma, x^+{:} \oplus \langle\, l_i : T_i \,\rangle_{1 \leqslant i \leqslant n} \vdash_X x \triangleleft l.P{:}Y} \text{TC-Choose}_2$$

$$\frac{\Gamma, x^-{:}T_i \vdash_{X - \{x^+\}} P{:}Y \quad l = l_i \in \{l_1, \ldots, l_n\} \quad x^- \in X \cap Y}{\Gamma, x^-{:} \oplus \langle\, l_i : T_i \,\rangle_{1 \leqslant i \leqslant n} \vdash_X x \triangleleft l.P{:}Y} \text{TC-Choose}_3$$

In TC-InS$_i$, $Y^S = \{y_i \mid U_i \in \mathit{SType}\}$. In TC-OutS$_i$, $Y^S = \{y_i^{q_i} \mid U_i \in \mathit{SType}\}$.

Fig. 13 Inference rules for the typechecking algorithm, continued

Definition 16. *If Γ is an environment and X is a set of optionally polarized names, then $\Gamma|_X = \{x^p{:}T \in \Gamma \mid x^p \in X \text{ or } T \notin SType\}$.*

Lemma 16. *The following properties are invariants of the typechecking algorithm.*

1. *$Y \subseteq X \subseteq dom(\Gamma)$, and for every $x^p \in X$, $\Gamma(x^p) \in SType$.*
2. *for all names x, if $x^+ \in X$ or $x^- \in X$ then $x \notin X$.*
3. *$\Gamma|_X$ is balanced*
4. *if $\{x^+, x^-\} \subseteq X$ then $\Gamma(x^+) \neq \mathsf{end}$ and $\Gamma(x^-) \neq \mathsf{end}$*

Proof. The top-level call of the typechecker satisfies the invariants because Γ is balanced and $X = dom(\Gamma)$. We check, for each inference rule, that if the conclusion satisfies the invariants then so do the hypotheses. Property (1) is straightforward. Property (2) relies on the fact that the additions to X in rules TC-NewS, TC-In and TC-InS_i are bound names which can be assumed not to be in X. Property (3) relies on the fact that in rules TC-InS_i, TC-OutS_i, TC-Offer$_i$ and TC-Choose$_i$ (where the change in the type of x^p could potentially unbalance the environment), $x^{\overline{p}}$ is removed from X, so it is not possible that both x^+ and x^- are in the environment $\Gamma|_X$. Property (4) relies on the condition $T \neq \mathsf{end}$ in rule TC-NewS and the fact that in rules TC-InS_i, TC-OutS_i, TC-Offer$_i$ and TC-Choose$_i$ (where the type of x^p could become end), $x^{\overline{p}}$ is removed from X. □

We can now clarify the interpretation of the inference rules in Figures 12 and 13 as an algorithm; we describe each case. In all cases, recursive types are unfolded until their structure is exposed.

TC-Nil: The algorithm assumes that all of the end types in $\Gamma|_X$ are introduced at this leaf of the derivation tree.

TC-Par: First typecheck P to calculate the session channels Y which it uses. These channels are removed from Γ and from X before typechecking Q. This ensures that each session channel is only used by one parallel component of the system.

TC-Rep: Typecheck P in an environment which contains no session channels.

TC-New, TC-NewS: The type declaration attached to (νx) determines which rule should be used. TC-New is straightforward because no session channels are involved. TC-NewS adds x^+ and x^- to Γ and to X, and checks that P actually uses them; they are removed from Y in order to maintain the invariant that $Y \subseteq X$. The condition $\mathsf{end} \not\leqslant S$ is necessary in order to maintain Property 4 of Lemma 16; it is expressed in terms of subtyping so that cases such as $\mu X.\mathsf{end}$ are covered. This condition means that processes of the form $(\nu x{:}\mathsf{end})P$ cannot be typechecked, but this seems to be a harmless restriction.

TC-In, TC-InS_i: The type of x in the environment determines which rule to use. If we have $x{:}\widehat{\ }[\widetilde{T}]$ then TC-In applies. If we have $x{:}S$ for some session type S then, by the contrapositive of Lemma 16(2), only TC-InS_1 applies; if S is not an input type then this is a type error. If both x^+ and x^- are in the environment then Lemma 16(3) guarantees that at most one of them has an input type, and this determines whether TC-InS_2 or TC-InS_3 applies (or neither, which is a type error). In a similar way to TC-NewS, P must actually use the session channels which are in $\widetilde{y}$; also, channel x must be present in X. The additional condition that x is in Y ensures that either x is used by P, in order to fully use its communication capabilities, or that P is $\mathbf{0}$ and x, if it is a session channel, has type end. In the case of TC-InS_2 or TC-InS_3, the opposite polarity of x is removed from X to ensure that it is not used by P; this is necessary in order to maintain Property 3 of Lemma 16, and corresponds to the discussion of typechecking at the beginning of this section.

TC-Out, TC-OutS_i: The choice between these rules is determined in the same way as for the input rules, and channel x is handled in the same way. In all cases, any session channels which form part of the message must be removed from X before typechecking P.

TC-Offer$_i$: The choice between these rules is determined in the same way as the previous cases. The Y calculated by typechecking each P_i must all be the same; this requires all of the P_i to use exactly the same set of session channels, and corresponds to the condition in rule T-Offer that all branches are typed in the same environment.

TC-Choose$_i$: The choice of rule is determined in the usual way. The rules themselves behave like simplified forms of the TC-OutS rules.

We now prove that successful typechecking of a process implies the existence of a typing in the original system, and hence that execution is safe.

Definition 17. *The function Erase removes all polarities from the names within a process.*

Theorem 5 (Soundness of Typechecking). *If* $\Gamma \vdash_X \mathit{Erase}(P){:}Y$ *then* $\Gamma|_Y \vdash P$.

Proof. By induction on the derivation of $\Gamma \vdash_X \mathit{Erase}(P){:}Y$, with a case-analysis on the last rule used.

TC-Nil: As $\mathit{Erase}(\mathbf{0}) = \mathbf{0}$, we have $\Gamma \vdash_X \mathbf{0}{:}X$ and $\Gamma|_X$ is completed. Rule T-Nil gives $\Gamma|_X \vdash \mathbf{0}$ directly.

TC-Par: As $\mathit{Erase}(P \mid Q) = \mathit{Erase}(P) \mid \mathit{Erase}(Q)$, we have $\Gamma \vdash_X P|Q{:}Y \cup Z$ where the hypotheses of the rule are $\Gamma \vdash_X \mathit{Erase}(P){:}Y$ and $\Gamma - Y \vdash_{X-Y} \mathit{Erase}(Q){:}Z$. The induction hypothesis gives $\Gamma|_Y \vdash P$

and $(\Gamma - Y)|_Z \vdash Q$. Rule T-Par gives $\Gamma|_Y + (\Gamma - Y)|_Z \vdash P \mid Q$ (the sum of environments is defined because they have no session channels in common). Finally, $\Gamma|_Y + (\Gamma - Y)|_Z = \Gamma|_{Y \cup Z}$.

TC-New: As $\mathit{Erase}((\nu x{:}T)P) = (\nu x{:}T)\mathit{Erase}(P)$, we have $\Gamma \vdash_X (\nu x{:}T)\mathit{Erase}(P){:}Y$ where the hypothesis of the rule is $\Gamma, x{:}T \vdash_X \mathit{Erase}(P){:}Y$. The induction hypothesis gives $(\Gamma, x{:}T)|_Y \vdash P$. Because $(\Gamma, x{:}T)|_Y = \Gamma|_Y, x{:}T$, rule T-New gives $\Gamma|_Y \vdash (\nu x{:}T)P$.

TC-NewS: As $\mathit{Erase}((\nu x{:}S)P) = (\nu x{:}S)\mathit{Erase}(P)$, we have $\Gamma \vdash_X (\nu x{:}S)\mathit{Erase}(P){:}Y - \{x^+, x^-\}$ where the main hypothesis of the rule is $\Gamma, x^+{:}S, x^-{:}\overline{S} \vdash_{X \cup \{x^+, x^-\}} \mathit{Erase}(P){:}Y$. The induction hypothesis gives $(\Gamma, x^+{:}S, x^-{:}\overline{S})|_Y \vdash P$, so $\Gamma|_Y, x^+{:}S, x^-{:}\overline{S} \vdash P$ because $\{x^+, x^-\} \subseteq Y$. Because $S \perp_c \overline{S}$ by Proposition 5, rule T-NewS gives $\Gamma|_Y \vdash (\nu x{:}S)P$.

TC-In: Because $\mathit{Erase}(x?[\widetilde{y}{:}\widetilde{U}].P) = x?[\widetilde{y}{:}\widetilde{U}].\mathit{Erase}(P)$, we have $\Gamma, x{:}\widehat{\ }[\widetilde{T}] \vdash_X x?[\widetilde{y}{:}\widetilde{U}].\mathit{Erase}(P){:}Y - \{y_i \mid U_i \in \mathit{SType}\}$ where the main hypothesis of the rule is $\Gamma, x{:}\widehat{\ }[\widetilde{T}], \widetilde{y}{:}\widetilde{U} \vdash_{X \cup \{y_i \mid U_i \in \mathit{SType}\}} \mathit{Erase}(P){:}Y$. The induction hypothesis gives $(\Gamma, x{:}\widehat{\ }[\widetilde{T}], \widetilde{y}{:}\widetilde{U})|_Y \vdash P$. The restriction of the environment to Y removes the y_i such that U_i is not a session type, but by Lemma 6 we can re-introduce them to obtain $\Gamma|_Y, x{:}\widehat{\ }[\widetilde{T}], \widetilde{y}{:}\widetilde{U} \vdash P$. Rule T-In gives $\Gamma|_Y, x{:}\widehat{\ }[\widetilde{T}] \vdash x?[\widetilde{y}{:}\widetilde{U}].P$, which is what we need because $(\Gamma, x{:}\widehat{\ }[\widetilde{T}])|_{Y - \{y_i \mid U_i \in \mathit{SType}\}} = \Gamma|_Y, x{:}\widehat{\ }[\widetilde{T}]$.

TC-Out: Because $\mathit{Erase}(x![\widetilde{y}^{\widetilde{q}}].P) = x![\widetilde{y}].\mathit{Erase}(P)$, we have $\Gamma, x{:}\widehat{\ }[\widetilde{T}] \vdash_X x![\widetilde{y}].\mathit{Erase}(P){:}Y \cup \{\widetilde{y}^{\widetilde{q}}\}$ where the main hypothesis of the rule is $(\Gamma, x{:}\widehat{\ }[\widetilde{T}]) - \widetilde{y}^{\widetilde{q}}{:}\widetilde{U} \vdash_{X - \{y_i^{q_i} \mid U_i \in \mathit{SType}\}} \mathit{Erase}(P){:}Y$. The polarities $\widetilde{q}$ for $\widetilde{y}$ can be calculated as follows. The construction of $(\Gamma, x{:}\widehat{\ }[\widetilde{T}]) - \widetilde{y}^{\widetilde{q}}$ implicitly requires a check that $\widetilde{y}^{\widetilde{q}}{:}\widetilde{U} \in \Gamma$ for some optional polarities $\widetilde{q}$ and types $\widetilde{U}$. If, for some i, both y_i^+ and y_i^- are in the environment, then by Properties 3 and 4 of Lemma 16, their types are dual and not equal to end. Therefore by Lemma 3 at most one of them can be a subtype of T_i. The induction hypothesis gives $((\Gamma, x{:}\widehat{\ }[\widetilde{T}]) - \widetilde{y}^{\widetilde{q}}{:}\widetilde{U})|_Y \vdash P$ and so rule T-Out gives $((\Gamma, x{:}\widehat{\ }[\widetilde{T}]) - \widetilde{y}^{\widetilde{q}}{:}\widetilde{U})|_Y + \widetilde{y}^{\widetilde{q}} \vdash x![\widetilde{y}^{\widetilde{q}}].P$. Because $((\Gamma, x{:}\widehat{\ }[\widetilde{T}]) - \widetilde{y}^{\widetilde{q}}{:}\widetilde{U})|_Y + \widetilde{y}^{\widetilde{q}} = (\Gamma, x{:}\widehat{\ }[\widetilde{T}])|_{Y \cup \{\widetilde{y}^{\widetilde{q}}\}}$, this is the required judgement.

TC-InS$_i$: These three cases are essentially the same as TC-In. The condition $x \in Y$ guarantees that x is present in $(\Gamma, x{:}S, \widetilde{y}{:}\widetilde{U})|_Y$.

TC-OutS$_i$: These three cases are essentially the same as TC-Out. The condition $x \in Y$ has the same purpose as for TC-InS$_i$.

TC-Offer$_i$: Consider TC-Offer$_1$; the others use the same reasoning. Because $\mathit{Erase}(x \triangleright \{\ l_i : P_i\ \}_{1 \leqslant i \leqslant n}) = x \triangleright \{\ l_i : \mathit{Erase}(P_i)\ \}_{1 \leqslant i \leqslant n}$, we have $\Gamma, x{:}\&\langle\, l_i : T_i \,\rangle_{1 \leqslant i \leqslant m} \vdash_X x \triangleright \{\ l_i : \mathit{Erase}(P_i)\ \}_{1 \leqslant i \leqslant n}{:}Y$ with $\Gamma, x{:}T_i \vdash_X \mathit{Erase}(P_i){:}Y$ as hypothesis i of the rule. The induction

hypothesis gives $(\Gamma, x{:}T_i)|_Y \vdash P_i$ for each i. Because $x \in Y$ we have $(\Gamma, x{:}T_i)|_Y = \Gamma|_Y, x{:}T_i$, and so rule T-Offer gives $\Gamma|_Y, x{:}\&\langle\, l_i : T_i \,\rangle_{1 \leqslant i \leqslant m} \vdash_X x \triangleright \{\, l_i : P_i \,\}_{1 \leqslant i \leqslant n}$, which is the required typing because $(\Gamma, x{:}\&\langle\, l_i : T_i \,\rangle_{1 \leqslant i \leqslant m})|_Y = \Gamma|_Y, x{:}\&\langle\, l_i : T_i \,\rangle_{1 \leqslant i \leqslant m}$.

TC-Choose$_i$: These cases use similar reasoning to TC-Offer$_i$ and TC-OutS_i. □

6 Conclusions

We have added a notion of subtyping to a system of session types for the pi calculus, formalized the syntax, operational semantics and typing rules of the resulting language, and proved that a correctly-typed process executes without communication errors. We have demonstrated that subtyping increases the flexibility of session types as specifications of protocols in, for example, client-server systems. We have also shown that our typing rules, presented declaratively for formal convenience, can be converted into a practical typechecking algorithm which is also able to infer the polarities of channels.

The most obvious direction for future work is to study behavioural equivalence in the presence of session types. It is likely that adding assumptions about typability to a standard bisimulation equivalence will provide stronger reasoning principles for process equivalence; this effect has been found in other type systems for the pi calculus [18, 25, 27]. Another possibility is to investigate polymorphism in session types. We have some preliminary results on a form of bounded polymorphism [13], but we have not yet considered the full range of possibilities.

Acknowledgements

This research was funded by the EPSRC project "Novel Type Systems for Concurrent Programming Languages" (grants GR/L75177, GR/N39494). The first author would like to thank the anonymous referees for their detailed and insightful comments.

References

1. E. Bonelli, A. Compagnoni, and E. Gunter. Correspondence assertions for process synchronization in concurrent communication. *Journal of Functional Programming*, 15(2):219–247, 2005.

2. E. Bonelli, A. Compagnoni, and E. Gunter. Typechecking safe process synchronization. In *Proceedings of the Third EATCS Workshop on the Foundations of Global Ubiquitous Computing*, Electronic Notes on Theoretical Computer Science. Elsevier, 2005. To appear.
3. S. Chaki, S. K. Rajamani, and J. Rehof. Types as models: model checking message-passing programs. In *Proceedings, 29th ACM Symposium on Principles of Programming Languages*, pages 45–57. ACM Press, 2002.
4. R. DeLine and M. Fähndrich. Enforcing high-level protocols in low-level software. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI) (SIGPLAN Notices 36(5))*, pages 59–69. ACM Press, 2001.
5. M. Dezani-Ciancaglini, N. Yoshida, A. Ahern, and S. Drossopolou. A distributed object-oriented language with session types. In *Proceedings of the Symposium on Trustworthy Global Computing*. Springer, 2005. To appear.
6. S. J. Gay. A sort inference algorithm for the polyadic π-calculus. In *Proceedings, 20th ACM Symposium on Principles of Programming Languages*. ACM Press, 1993.
7. S. J. Gay and M. J. Hole. Types and subtypes for client-server interactions. In *ESOP'99: Proceedings of the European Symposium on Programming Languages and Systems*, volume 1576 of *Lecture Notes in Computer Science*, pages 74–90. Springer, 1999.
8. S. J. Gay and M. J. Hole. Types and subtypes for correct communication in client-server systems. Technical Report TR-2003-131, Department of Computing Science, University of Glasgow, February 2003.
9. J.-Y. Girard. Linear Logic. *Theoretical Computer Science*, 50(1):1–102, 1987.
10. A. D. Gordon and A. Jeffrey. Typing correspondence assertions for communication protocols. *Theoretical Computer Science*, 300(1–3):379–409, 2003.
11. D. Grossman. Type-safe multithreading in cyclone. In *Proceedings of the ACM Workshop on Types in Language Design and Implementation (SIGPLAN Notices 38(3))*, pages 13–25. ACM Press, 2003.
12. D. Grossman, G. Morrisett, T. Jim, M. Hicks, Y. Wang, and J. Cheney. Region-based memory management in cyclone. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI) (SIGPLAN Notices 37(5))*, pages 282–293. ACM Press, 2002.
13. M. J. Hole and S. J. Gay. Bounded polymorphism in session types. Technical Report TR-2003-132, Department of Computing Science, University of Glasgow, March 2003.
14. K. Honda. Types for dyadic interaction. In *CONCUR'93: Proceedings of the International Conference on Concurrency Theory*, volume 715 of *Lecture Notes in Computer Science*, pages 509–523. Springer, 1993.
15. K. Honda, V. Vasconcelos, and M. Kubo. Language primitives and type discipline for structured communication-based programming. In *ESOP'98: Proceedings of the European Symposium on Programming*, volume 1381 of *Lecture Notes in Computer Science*, pages 122–138. Springer-Verlag, 1998.
16. A. Igarashi and N. Kobayashi. A generic type system for the pi-calculus. *Theoretical Computer Science*, 311(1–3):121–163, 2004.
17. N. Kobayashi. A partially deadlock-free typed process calculus. *ACM Transactions on Programming Languages and Systems*, 20:436–482, 1998.
18. N. Kobayashi, B. C. Pierce, and D. N. Turner. Linearity and the Pi-Calculus. *ACM Transactions on Programming Languages and Systems*, 21(5):914–947, September 1999.

19. I. Mackie. Lilac : A functional programming language based on linear logic. *Journal of Functional Programming*, 4(4):1–39, October 1994.
20. R. Milner. The polyadic π-calculus: A tutorial. Technical Report 91-180, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, 1991.
21. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I and II. *Information and Computation*, 100(1):1–77, September 1992.
22. J. Myers and M. Rose. Post office protocol version 3, May 1996. Internet Standards RFC1939.
23. M. Neubauer and P. Thiemann. An implementation of session types. In *Practical Aspects of Declarative Languages (PADL'04)*, volume 3057 of *Lecture Notes in Computer Science*, pages 56–70. Springer, 2004.
24. M. Neubauer and P. Thiemann. Protocol specialization. In *Proceedings of the Second Asian Symposium on Programming Languages and Systems (APLAS 2004)*, volume 3302 of *Lecture Notes in Computer Science*, pages 246–261. Springer, 2004.
25. B. C. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5), 1996.
26. B. C. Pierce. *Types and Programming Languages*. MIT Press, 2002.
27. B. C. Pierce and D. Sangiorgi. Behavioral equivalence in the polymorphic pi-calculus. *Journal of the ACM*, 47(3), 2000.
28. B. C. Pierce and D. N. Turner. Pict: A programming language based on the pi-calculus. In G. Plotkin, C. Stirling, and M. Tofte, editors, *Proof, Language and Interaction: Essays in Honour of Robin Milner*. MIT Press, 2000.
29. S. K. Rajamani and J. Rehof. A behavioral module system for the pi-calculus. In *Static Analysis: 8th International Symposium, SAS 2001*, volume 2126 of *Lecture Notes in Computer Science*, pages 375–394. Springer, 2001.
30. D. Sangiorgi and D. Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.
31. K. Takeuchi, K. Honda, and M. Kubo. An interaction-based language and its typing system. In *PARLE '94: Parallel Architectures and Languages Europe*, volume 817 of *Lecture Notes in Computer Science*. Springer, 1994.
32. D. N. Turner. *The Polymorphic Pi-Calculus: Theory and Implementation*. PhD thesis, University of Edinburgh, 1996.
33. A. Vallecillo, V. T. Vasconcelos, and A. Ravara. Typing the behavior of objects and components using session types. In *International Workshop on Foundations of Coordination Languages and Software Architectures*, volume 68(3) of *Electronic Notes in Theoretical Computer Science*. Elsevier, 2003.
34. V. T. Vasconcelos and K. Honda. Principal typing schemes in a polyadic π-calculus. In *CONCUR'93: Proceedings of the International Conference on Concurrency Theory*, volume 715 of *Lecture Notes in Computer Science*. Springer, 1993.
35. V. T. Vasconcelos, A. Ravara, and S. J. Gay. Session types for functional multithreading. In *CONCUR'04: Proceedings of the International Conference on Concurrency Theory*, volume 3170 of *Lecture Notes in Computer Science*, pages 497–511. Springer, 2004. Full version to appear in *Theoretical Computer Science*.