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

Bounded polymorphism in session types

Simon J. Gay

Department of Computing Science, University of Glasgow, UK
Email: `simon@dcs.gla.ac.uk`

Received 9 December 2006

Session types allow high-level specifications of structured patterns of communication, such as client-server protocols, to be expressed as types and verified by static typechecking. In collaboration with Malcolm Hole, we have previously introduced a notion of subtyping for session types, formulated for an extended pi calculus. Subtyping allows one part of a system, for example a server, to be refined without invalidating type-correctness of other parts, for example clients. We now introduce bounded polymorphism, based on the same notion of subtyping, in order to support more precise and flexible specifications of protocols; in particular, a choice of type in one message may affect the types of future messages. We formalize the syntax, operational semantics and typing rules of an extended pi calculus, and prove that typechecking guarantees absence of run-time communication errors. We study algorithms for checking instances of the subtype relation in two versions of our system, which we call Kernel $S_{\leqslant}$ and Full $S_{\leqslant}$, and establish that subtyping in Kernel $S_{\leqslant}$ is decidable but subtyping in Full $S_{\leqslant}$ is undecidable.

1. Introduction

Distributed systems are typically structured around protocols which specify the form and sequence of communications between agents. Such protocols are often complex, involving substantial numbers of states and a variety of state transitions caused by different types of message. When implementing an agent which is intended to follow a particular protocol, it is desirable to be able to verify (preferably automatically) that the sequence and structure of messages sent and received is correct according to the protocol. However, standard programming languages do not provide good support for this kind of verification.

The theory of *session types* addresses this problem by defining a notion of type which can capture the specification of a protocol. Session types can be associated with communication channels, and the actual use of a channel by a program can be statically checked against its type. For example, in a client-server system, the type $?[\mathsf{int}].![\mathsf{int}].\mathsf{end}$ on the server side of a channel specifies a simple protocol in which the server receives an integer and sends back an integer; the type

$$S_1 = \&\langle\, \mathsf{go}\!:?[\mathsf{int}].![\mathsf{int}].\mathsf{end},\ \mathsf{quit}\!:\mathsf{end}\,\rangle \qquad (1)$$

specifies a choice (&) between immediate termination (quit) and non-trivial behaviour (go). Session types were first proposed by (Honda 1993) and have been further developed by a number of researchers including the present author (Bonelli et al. 2005; Fähndrich et al. 2006; Garralda et al. 2006; Gay and Hole 1999, 2005; Takeuchi et al. 1994; Honda et al. 1998; Neubauer and Thiemann 2004a,b; Vallecillo et al. 2006; Vasconcelos et al. 2004, 2006; Yoshida and Vasconcelos 2006). Recently, session types have begun to be applied to the specification of web services and business processes (Carbone et al. 2006).

A session type describes a bilateral or two-party protocol. Throughout this paper we will use simple client-server systems as examples, but the applications of session types are not restricted to this case; in general they can describe any bilateral protocol used within a distributed system. Given that a protocol specifies which party sends the first message, it is convenient to view the initial sender as a client and the initial receiver as a server, but this classification applies only to that particular protocol run. In general, an agent within a distributed system may, at different times, find itself on the server side or the client side of various protocols, and it may even be interleaving runs of several protocols, acting as client or server independently in each one.

In previous papers (Gay and Hole 1999, 2005) we increased the expressive power of session types by defining a notion of subtyping. The main application was to allow server upgrades, which alter the protocol and hence its session type, to be made without removing the type-correctness of older clients. If the specification (1) above is modified to

$$S_2 = \&\langle\, \mathsf{go}\colon ?[\mathsf{int}].![\mathsf{int}].\mathsf{end},\ \mathsf{new}\colon ?[\mathsf{real}].![\mathsf{bool}].\mathsf{end},\ \mathsf{quit}\colon\mathsf{end}\,\rangle \tag{2}$$

then we have $S_1 \leqslant S_2$, meaning that a client that only knows about go and quit can be successfully typechecked against S_2. Subtyping is explained further in Section 2.3.

We are now interested in polymorphic protocol specifications. We naturally have subtype polymorphism: for example if $\mathsf{int} \leqslant \mathsf{real}$ then a client of protocol (2) can send an int or a real after selecting new. However, because input (?) behaves covariantly and output (!) behaves contravariantly, the type

$$S_3 = \&\langle\, \mathsf{go}\colon ?[\mathsf{real}].![\mathsf{real}].\mathsf{end},\ \mathsf{quit}\colon\mathsf{end}\,\rangle. \tag{3}$$

cannot be used to specify a protocol in which the go service receives and sends values of the same subtype of real. This situation is familiar from object-oriented languages, if we think of receiving a method parameter and sending the result.

In this paper we introduce a notion of bounded polymorphism, similar in general terms to $\mathrm{F}_{<:}$(Cardelli and Wegner 1985). We choose to associate polymorphism with the labels in the branching (&) type; for example, instead of (3) we define

$$S_4 = \&\langle\, \mathsf{go}(X \leqslant \mathsf{real})\colon ?[X].![X].\mathsf{end},\ \mathsf{quit}\colon\mathsf{end}\,\rangle$$

and allow the client, when choosing go, to instantiate X to either int or real. As far as we know this is the first study of bounded polymorphism in the pi calculus, and the first study of any form of polymorphism in relation to session types. In addition to the example above, and other examples in Section 2, recent interest in session types for object-oriented languages (Dezani-Ciancaglini et al. 2005, 2006) means that bounded polymorphism may become relevant to integrating session types with a language such as Java 1.5.

We expand on the above explanation in Section 2, then formalize our system in Section 3 and prove a type safety result in Section 4. In Section 5 we discuss practical typechecking and prove that our subtype relation is decidable. In Section 6 we generalize our subtype relation in a similar way to the generalization from Kernel $F_{<:}$to Full $F_{<:}$, and prove that this makes the subtype relation undecidable. Finally, in Section 7, we discuss related work as well as possible extensions to our language.

2. Session types and bounded polymorphism

2.1. *Review of session types*

Session types in the pi calculus describe structured bilateral interaction between agents in a distributed system. Both parties participating in a sequence of communications—a protocol run—are statically type checked, not only to ensure that the types of messages sent are those expected, but also that the ordering of messages is correct. Furthermore, choices can be offered and selections made by both parties, the whole dialogue taking place on a single channel. The examples in this section are based on simple client-server systems; more generally, in any bilateral protocol, we can view the agent that sends the first message as the client and the agent that receives the first message as the server.

Consider a server for mathematical operations which offers sine and square functions. The communications with the client take place on a single session channel, x, with ports x^+ and x^-. We call $^+$ and $^-$ *polarities*, and we use them to indicate which end of a session channel is being used in each occurrence in a process. The need for polarities is explained in Section 2.2. In previous work (Gay and Hole 2005) we have shown that a modification of the top-level type system allows polarities to be inferred, but it is simpler to develop the theory if polarities are included and so we use them in the present paper.

In this example, the server will use x^+ and the client x^-. Each of x^+ and x^- has an associated session type, which we will call S_5 and $\overline{S_5}$, respectively.

$$\begin{array}{rcl} S_5 & = & \&\langle\, \mathsf{sin}:?[\mathsf{real}].![\mathsf{real}].\mathsf{end}, \mathsf{sqr}:?[\mathsf{int}].![\mathsf{int}].\mathsf{end}\,\rangle \\ \overline{S_5} & = & \oplus\langle\, \mathsf{sin}:![\mathsf{real}].?[\mathsf{real}].\mathsf{end}, \mathsf{sqr}:![\mathsf{int}].?[\mathsf{int}].\mathsf{end}\,\rangle \end{array}$$

The $\&\langle \dots \rangle$ constructor (*branch*) in the type S_5 indicates that a choice is offered, in this case between two labels, sin and sqr, each of which then has a continuation type representing a series of inputs and outputs (? for input, ! for output). Dually, the $\oplus\langle \dots \rangle$ constructor (*choice*) in the type $\overline{S_5}$ indicates the making of a choice. The pattern of sending and receiving for each label here is the opposite of that in the type of the server's port.

The server and a possible client, parameterized on the port, could be implemented by

$$\begin{array}{rcl} \mathsf{serverbody}_1(y) & = & y \rhd \{\ \mathsf{sin}:y?[a{:}\mathsf{real}].y![\mathsf{sin}(a)].\mathbf{0}, \\ & & \qquad \mathsf{sqr}:y?[a{:}\mathsf{int}].x![a^2].\mathbf{0}\ \} \\ \mathsf{clientbody}_1(z) & = & z \lhd \mathsf{sin}.z![90].z?[r{:}\mathsf{real}].\mathbf{0} \end{array}$$

Here the server uses the $\rhd\{\dots\}$ construct (*offer*) to offer the labels sin and sqr. Each label has a continuation process performing the necessary inputs and outputs as specified in the type. The client uses the $\triangleleft$ construct (*choose*) to choose the label sin from those on offer and then performs the appropriate sequence of outputs and inputs.

Our type system, defined formally in Section 3.4, allows the judgements

$$\begin{array}{r}x^+{:}S_5 \vdash \mathsf{serverbody}_1(x^+)\\ x^-{:}\overline{S_5} \vdash \mathsf{clientbody}_1(x^-)\end{array}$$

to be derived, checking that the communication structure within each process matches the structure of the session type S_5 or $\overline{S_5}$, respectively.

Combining the server and client in parallel, we can derive

$$x^+{:}S_5, x^-{:}\overline{S_5} \vdash \mathsf{serverbody}_1(x^+) \mid \mathsf{clientbody}_1(x^-)$$

and the operational semantics formalized in Section 3.2 defines the behaviour of this parallel combination 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. Our standard method is for the client to create a channel x of type S_5 and send one end of it, x^+, to the server along a standard (non-session-typed) channel a of type $\widehat{\ }[S_5]$. The system is defined and typed as follows.

$$\begin{array}{rcl}\mathsf{server} & = & a?[y{:}S_5].\mathsf{serverbody}_1(y)\\ \mathsf{client} & = & (\nu x{:}S_5)(a![x^+].\mathsf{clientbody}_1(x^-))\end{array}$$

$$a : \widehat{\ }[S_5] \vdash \mathsf{client} \mid \mathsf{server}$$

The system reduces to

$$(\nu x{:}S_5)(\mathsf{clientbody}_1(x^-) \mid \mathsf{serverbody}_1(x^+))$$

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: for example, after sending x^+ to server, client must not make any further use of it. This restriction is related to linear (Girard 1987) control of values and our type system treats session channels in a similar way to the linear and linearized channels studied by (Kobayashi et al. 1999). The difference is that each end of a session channel may be used many times by the process that owns it.

2.2. *Polarities*

The reason for using polarities to distinguish between the two ports of a channel is that while constructing a typing derivation there may be points at which the types of the two ports are not directly related (in particular, they are not dual to each other). An example is the following derivation, using the typing rules presented in Section 3.4:

$$\dfrac{\dfrac{\dfrac{\dfrac{x^+ : \mathsf{end}, x^- : \mathsf{end}, z : \mathsf{int}\ \text{completed}}{x^+ : \mathsf{end}, x^- : \mathsf{end}, z : \mathsf{int} \vdash \mathbf{0}}\ \textsc{T-Nil}}{x^+ : \mathsf{end}, x^- : ?[\mathsf{int}].\mathsf{end} \vdash x^-?[z{:}\mathsf{int}].\mathbf{0}}\ \textsc{T-InS}}{x^+ : ![\mathsf{int}].\mathsf{end}, x^- : ?[\mathsf{int}].\mathsf{end} \vdash x^+![2].x^-?[z{:}\mathsf{int}].\mathbf{0}}\ \textsc{T-OutS}}{\vdash (\nu x{:}![\mathsf{int}].\mathsf{end})x^+![2].x^-?[z{:}\mathsf{int}].\mathbf{0}}\ \textsc{T-New}$$

in which, on the third line, x^+ and x^- have different types. It would therefore not be possible to refer to both of them simply as x.

According to the operational semantics defined in Section 3.2 the process

$$(\nu x{:}![\mathsf{int}].\mathsf{end})x^+![2].x^-?[z{:}\mathsf{int}].\mathbf{0}$$

is deadlocked: communication between the output on x^+ and the input on x^- could only happen if they were in separate parallel processes. One might therefore ask why we want this process to be typable at all. The answer is that it can arise during execution. For example, we have the following reduction sequence:

$$\begin{array}{c}
(\nu x{:}![\mathsf{int}].\mathsf{end})a![x^-].a?[y{:}![\mathsf{int}].\mathsf{end}].x^+![2].y?[z{:}\mathsf{int}].\mathbf{0} \quad | \quad a?[z{:}?[\mathsf{int}].\mathsf{end}].a![z].\mathbf{0} \\
\downarrow \\
(\nu x{:}![\mathsf{int}].\mathsf{end})a?[y{:}![\mathsf{int}].\mathsf{end}].x^+![2].x^-?[z{:}\mathsf{int}].\mathbf{0} \quad | \quad a![x^-].\mathbf{0} \\
\downarrow \\
(\nu x{:}![\mathsf{int}].\mathsf{end})x^+![2].x^-?[z{:}\mathsf{int}].\mathbf{0} \quad | \quad \mathbf{0}
\end{array}$$

and the initial process is typable in the environment $a : \ulcorner[?[\mathsf{int}].\mathsf{end}]$. In order to prove that typability is preserved by reduction (Section 4) we need to be able to type every step in the reduction sequence above, and so we use polarities in order to have a type system in which the final process has a typing derivation. Note that our type system does not aim to guarantee deadlock-freedom.

2.3. *Subtyping*

Continuing the example from Section 2.1, suppose that a tangent operation is added to the server. The new protocol is specified by

$$S_6 = \&\langle\, \mathsf{sin}: ?[\mathsf{real}].![\mathsf{real}].\mathsf{end}, \mathsf{sqr}: ?[\mathsf{int}].![\mathsf{int}].\mathsf{end}, \mathsf{tan}: ?[\mathsf{real}].![\mathsf{real}].\mathsf{end}\,\rangle$$

and the server implementation becomes

$$\begin{array}{rcl}
\mathsf{serverbody}_2(x) & = & x \triangleright \{\ \mathsf{sin}: x?[a{:}\mathsf{real}].x![\mathsf{sin}(a)].\mathbf{0}, \\
& & \quad\ \mathsf{sqr}: x?[a{:}\mathsf{real}].x![a^2].\mathbf{0}, \\
& & \quad\ \mathsf{tan}: x?[a{:}\mathsf{real}].x![\mathsf{tan}(a)].\mathbf{0}\ \} \\
\mathsf{server}_2 & = & a?[y{:}S_6].\mathsf{serverbody}'(y).
\end{array}$$

We have the typing judgements

$$\begin{array}{c}
x^+{:}S_6 \vdash \mathsf{serverbody}_2(x^+) \\
a : \ulcorner[S_6] \vdash \mathsf{server}_2.
\end{array}$$

According to the definition of subtyping for session types (Gay and Hole 1999, 2005), $S_5 \leqslant S_6$. It is not obvious why this is correct (rather than $S_6 \leqslant S_5$), but it can be understood in the following ways.

1 We want a system consisting of the original client and the new server to be typable, because the original client is still correctly using services of the new server. It should not matter that the original client does not know about the new service. We therefore want to be able to derive (note the change in the type of a)

$$a : \widehat{\ }[S_6] \vdash \mathsf{client}_1$$

and this is indeed derivable in our type system. The behaviour of client_1 is to send a value of type S_5 on a channel of type $\widehat{\ }[S_6]$. If $S_5 \leqslant S_6$ then this is exactly the same situation as in the system of subtyping for pi calculus defined by (Pierce and Sangiorgi 1993, 1996).

2 We have the typing

$$a : \widehat{\ }[S_5] \vdash \mathsf{client}_1 \mid \mathsf{server}_2$$

and in order for the first step of reduction to preserve typability we need

$$\emptyset \vdash (\nu x{:}S_5)(\mathsf{clientbody}_1(x^-) \mid \mathsf{serverbody}_2(x^+))$$

which in turn requires

$$x^+ : S_5, x^- : \overline{S_5} \vdash \mathsf{clientbody}_1(x^-) \mid \mathsf{serverbody}_2(x^+)$$

which requires

$$x^+ : S_5 \vdash \mathsf{serverbody}_2(x^+).$$

If $S_5 \leqslant S_6$ then this follows from

$$y : S_6 \vdash \mathsf{serverbody}_2(y)$$

by a substitution lemma of exactly the usual kind. For example, in a functional language in which $\mathsf{int} \leqslant \mathsf{real}$ we would expect $y : \mathsf{real} \vdash \mathsf{sin}(y) : \mathsf{real}$ to imply $x : \mathsf{int} \vdash \mathsf{sin}(x) : \mathsf{real}$.

3 Because a branch type looks like a record type, in the sense that it has a set of labelled components, the fact that subtyping for branch types is covariant in the set of labels may be counterintuitive; recall that subtyping for record types is contravariant in the set of labels. However, a choice type looks equally similar to a record type, and subtyping for choice types is contravariant in the set of labels.

4 We can regard a branch type as analogous to an input type, with an explicit set of possible values (the labels) for the message rather than a type specifying the set of possible values. Similarly, a choice type is analogous to an output type. From this point of view, the covariance of branch (in the set of labels) and the contravariance of choice exactly match the covariance of input and the contravariance of output, both in our system and in that of (Pierce and Sangiorgi 1993, 1996).

2.4. *Bounded polymorphism*

Suppose we want to extend the sqr service to real numbers:

$$S_7 \quad = \quad \&\langle\, \mathsf{sin} : ?[\mathsf{real}].![\mathsf{real}].\mathsf{end}, \mathsf{sqr} : ?[\mathsf{real}].![\mathsf{real}].\mathsf{end}\,\rangle.$$

Assuming that $\mathsf{int} \leqslant \mathsf{real}$, a client can request the square of an int but will still receive a real result. A client that expects to receive an int result will not typecheck against S_7. To allow more precise specifications of protocols, we introduce bounded polymorphism, associating it with branch and choice types. Each branch is quantified by a type variable with upper and lower bounds. We define the type of our server's channel by

$$S_8 \quad = \quad \&\langle\, \mathsf{sin} : ?[\mathsf{real}].![\mathsf{real}].\mathsf{end}, \mathsf{sqr}(\mathsf{int} \leqslant X \leqslant \mathsf{real}) : ?[X].![X].\mathsf{end}\,\rangle$$

and the type of the client's side is

$$\overline{S_8} \quad = \quad \oplus\langle\, \mathsf{sin} : ![\mathsf{real}].?[\mathsf{real}].\mathsf{end}, \mathsf{sqr}(\mathsf{int} \leqslant X \leqslant \mathsf{real}) : ![X].?[X].\mathsf{end}\,\rangle.$$

The appropriate lower bound for X is the least (in the subtype relation) type for which squaring makes sense; we assume that this is int. The implementation of the server is

$$\begin{aligned} \mathsf{serverbody}_3(x) \quad = \quad x \triangleright \{\ & \mathsf{sin} : x?[a{:}\mathsf{real}].x![\mathsf{sin}(a)].\mathbf{0}, \\ & \mathsf{sqr}(\mathsf{int} \leqslant X \leqslant \mathsf{real}) : x?[a{:}X].x![a^2].\mathbf{0}\ \}. \end{aligned}$$

In order to typecheck

$$x{:}S_8 \vdash \mathsf{serverbody}_3(x)$$

it is necessary to check

$$\mathsf{int} \leqslant X \leqslant \mathsf{real}\,; x{:}?[X].![X].\mathsf{end} \vdash x?[a{:}X].x![a^2].\mathbf{0}$$

and for this we assume a type system for data expressions in which

$$\mathsf{int} \leqslant X \leqslant \mathsf{real}\,; a{:}X \vdash a^2{:}X$$

is derivable.

A client which uses the sqr service at type int is defined by

$$\mathsf{clientbody}_2(x) \quad = \quad x \triangleleft \mathsf{sqr}(\mathsf{int}).x![5].x?[r{:}\mathsf{int}].\mathbf{0}$$

and the code following $x \triangleleft \mathsf{sqr}(\mathsf{int})$ is typechecked with respect to the appropriate instantiation of $![X].?[X].\mathsf{end}$:

$$x{:}![\mathsf{int}].?[\mathsf{int}].\mathsf{end} \vdash x![5].x?[r{:}\mathsf{int}].\mathbf{0}$$

The complete system, after the initial creation of channel x and transmission of the server's end x^+, has the following reduction sequence, in which the annotations $x, \mathsf{sqr}(\mathsf{int})$

etc. show the channel and label (if any) involved in each reduction.

$$\begin{array}{rcl}
\mathsf{serverbody}_3(x^+) & \mid & \mathsf{clientbody}_2(x^-) \\
& \downarrow & {\scriptstyle x,\,\mathsf{sqr(int)}} \\
x^+?[a{:}\mathsf{int}].x^+![a^2].\mathbf{0} & \mid & x^-![5].x^-?[r{:}\mathsf{int}].\mathbf{0} \\
& \downarrow & {\scriptstyle x} \\
x^+![25].\mathbf{0} & \mid & x^-?[r{:}\mathsf{int}].\mathbf{0} \\
& \downarrow & {\scriptstyle x} \\
\mathbf{0} & \mid & \mathbf{0}
\end{array}$$

For notational simplicity, the formalization presented in this paper requires every label in a branch or choice type to specify exactly one bounded type variable. This could easily be generalized. Alternatively, an unquantified label such as sin in the example can be regarded as an abbreviation of $\mathsf{sin}(\mathsf{Bot} \leqslant X \leqslant \mathsf{Top})$ where X is a dummy type variable; multiple quantification such as

$$\mathsf{service}(T \leqslant X \leqslant T', U \leqslant Y \leqslant U') : S$$

can be encoded by introducing a nested branch:

$$\mathsf{service}(T \leqslant X \leqslant T') : \&\langle\, \mathsf{s}(U \leqslant Y \leqslant U') : S\,\rangle.$$

In some situations there might be a natural upper bound for a type variable but no natural lower bound, or vice versa. In this case the type Bot or Top can be used for the missing bound, as a kind of dummy in order to maintain a uniform syntax. As we will see later, the types Bot and Top cannot be used to instantiate polymorphism.

Associating bounded polymorphism with the labels in branch and choice types is not the only possibility. However, it seems reasonable that the common situation in a protocol is to have a choice between several options, each with its own sequence of typed messages and therefore its own possibility for instantiating polymorphism. The most obvious alternative would be to introduce a completely new construct allowing bounded quantification over a type variable, and a corresponding new kind of message which would instantiate a type variable and do nothing else. With this approach, it would never be necessary to introduce a branch type purely in order to introduce polymorphism; however, it would be necessary to introduce an extra message into every polymorphic branch of a branch type. On the assumption that polymorphism in protocols will usually coincide with a choice between two or more options, we feel that our approach is simpler.

2.5. *Session types as bounds*

We are not restricted to data types as bounds for type variables. A branching type can be moved up the subtype relation by extending the set of labels and/or moving the continuation types up the subtype relation; see Section 3.3 and (Gay and Hole 2005) for details. For example, if

$$S_9 = \&\langle\, \mathsf{sin} : ?[\mathsf{real}].![\mathsf{real}].\mathsf{end}, \mathsf{cos} : ?[\mathsf{real}].![\mathsf{real}].\mathsf{end}, \mathsf{sqr} : ?[\mathsf{int}].![\mathsf{int}].\mathsf{end}\,\rangle$$

then $S_5 \leqslant S_9$. Now consider a "testing server" which enables a client of a mathematical server to have the server tested before using it. The testing server receives the channel that the client intends to use to communicate with the mathematical server, carries out some tests, then returns a boolean result to the client, followed by the channel[†]. If the testing server is able to test the sin and sqr services specified by type S_5, then an appropriate type for communication with the testing server is

$$\&\langle\, \mathsf{test}(\mathsf{Bot} \leqslant X \leqslant \overline{S_5}) : ?[X].![\mathsf{bool}].![X].\mathsf{end}\,\rangle$$

where Bot is a dummy bound as explained in Section 2.4. A client of the testing server can give it a channel whose type is any subtype of $\overline{S_5}$; for example, because $S_5 \leqslant S_9$ we have $\overline{S_9} \leqslant \overline{S_5}$ (Lemma 4, Section 4). This means that the channel could be of a type which contains additional or refined services, but only sin and sqr will be tested.

It is very important to realize that, in our system, polymorphism cannot be instantiated with type Bot or Top. These types are used as bounds only to maintain a uniform syntax for the introduction of polymorphism. Thus, in the example above, the type variable X can only be instantiated by a type which actually is a choice type and therefore it will contain at least the labels sin and sqr.

2.6. *Lower and upper bounds*

The reason for including lower as well as upper bounds on type variables is so that type variables and their duals can be treated symmetrically. Suppose that we have $T \leqslant X \leqslant U$ in a branch label. It is possible to use $\overline{X}$ within the corresponding branch. If a channel of type $\overline{X}$ needs to be sent on another channel, of type $\widehat{\ }[V]$ say, then we need $\overline{X} \leqslant V$. Because duality reverses the subtype relation (Lemma 4, Section 4), this can only be estalished by making use of the lower bound for X: $T \leqslant X$ implies $\overline{X} \leqslant \overline{T}$ and by checking that $\overline{T} \leqslant V$, transitivity gives $\overline{X} \leqslant V$. It is not easy to find a natural example in which the lower bound must be used in this way, and it might turn out that in practice it is reasonable to prevent the use of $\overline{X}$ and work with upper bounds only. However, we have chosen to develop the theory in its symmetrical form.

A concrete, although artificial, example of the phenomenon described above is the following typing.

$$\begin{array}{l} x{:}\&\langle a(?[\mathsf{int}].\mathsf{end} \leqslant X \leqslant ?[\mathsf{real}].\mathsf{end}) : \mathsf{end}\rangle, z{:}\widehat{\ }[![\mathsf{int}].\mathsf{end}] \vdash \\ \qquad\qquad x \triangleright \{a(?[\mathsf{int}].\mathsf{end} \leqslant X \leqslant ?[\mathsf{real}].\mathsf{end}).(\nu y{:}X) z![y^-].y^+?[u{:}\mathsf{real}].\mathbf{0}\} \end{array}$$

Here the type of y^- is $\overline{X}$, so the output $z![y^-]$ requires $\overline{X} \leqslant ![\mathsf{int}].\mathsf{end}$. The input $y^+?[u{:}\mathsf{real}]$ does not play a significant role in illustrating this point; it is present so that there is a complete sequence of communications on y^+.

2.7. *Kernel and full versions of $S_\leqslant$*

We propose two versions of bounded polymorphism for session types, called Kernel $S_\leqslant$ and Full $S_\leqslant$ by analogy with Kernel and Full $F_{<:}$(Cardelli and Wegner 1985; Pierce

[†] Returning the channel is only worthwhile if the session type is made recursive, so that the same channel can be used for more requests.

2002). The difference is that in Kernel $S_{\leqslant}$ the bounds of quantified type variables do not change when moving up the subtype relation, whereas in Full $S_{\leqslant}$ the bounds are allowed to vary. The following example illustrates a situation in which the increased flexibility of Full $S_{\leqslant}$ is needed.

Consider a mathematical server which provides the service of addition on any subtype of real. It uses a session channel of type

$$S_{10} \quad = \quad \&\langle\, \mathsf{plus}(\mathsf{int} \leqslant X \leqslant \mathsf{real}) : ?[X].?[X].![X].\mathsf{end}\,\rangle.$$

If the server is upgraded to work with complex numbers (assuming that $\mathsf{real} \leqslant \mathsf{complex}$) then its session type becomes

$$S_{11} \quad = \quad \&\langle\, \mathsf{plus}(\mathsf{int} \leqslant X \leqslant \mathsf{complex}) : ?[X].?[X].![X].\mathsf{end}\,\rangle.$$

If an old client is to work with the new server then, exactly as in Section 3.3, we require $S_{10} \leqslant S_{11}$, which in turn requires variation of the upper bound of X.

Sections 3, 4 and 5 deal with Kernel $S_{\leqslant}$. Section 6 defines Full $S_{\leqslant}$.

2.8. *Related work*

Polymorphism in the style of the polymorphic lambda calculus (System F) (Girard 1972; Reynolds 1974) has previously been studied in the pi calculus by (Turner 1996) and (Pierce and Sangiorgi 2000), and implemented in the programming language Pict (Pierce and Turner 2000). That approach to polymorphism is based on existential quantification, as polymorphism is instantiated by a message consisting of a type and values of the same type, and the scope of a type variable is a single message. Our approach is based on universal quantification, and the scope of a type variable is an entire branch of communication, potentially including messages in both directions.

Weaker ML-style polymorphism has been studied in the pi calculus (or a related language) by (Vasconcelos and Honda 1993) and (Gay 1993). A rather different style of polymorphism has been proposed by (Liu and Walker 1995). It would be possible to formulate an ML-style polymorphic type system for session types, but this has not yet been done. The present paper is the first study of any kind of polymorphism for session types.

The question of decidability of subtyping in our systems is clearly related to decidability in $\mathrm{F}_{<:}$. Our proof of decidability for Kernel $S_{\leqslant}$ is closely based on the corresponding proof for Kernel $\mathrm{F}_{<:}$(Pierce 2002)[Chapter 28]. Our proof of undecidability for Full $S_{\leqslant}$ is based on an encoding of subtyping problems from Full $\mathrm{F}_{<:}$.

3. The language: syntax, semantics, type system

3.1. *Syntax*

Our language is based on monadic pi calculus with output prefixing (Milner et al. 1992; Sangiorgi and Walker 2001) and is very similar syntactically to the language proposed by (Gay and Hole 1999, 2005) for session types with subtyping. Synchronous output (i.e.

output as a prefix) simplifies the theory of session types by ensuring that outputs on a particular channel are unambiguously ordered at run-time. The significant modification of the syntax is that bounded polymorphic definitions are associated with the branching construct, and sending a label to make a choice also carries a specific type which instantiates the polymorphic definition. The restriction to a monadic syntax (i.e. every message is a single value rather than a tuple) is for notational convenience; the generalization to polyadic messages is best handled by introducing an expression language which includes product types. This issue is separate from the issues of bounded polymorphism which are the point of this paper. In any case, in a session type, a polyadic message can always be replaced by a sequence of monadic messages: for example, instead of $![\mathsf{int}, \mathsf{real}].S$ we can consider $![\mathsf{int}].![\mathsf{real}].S$.

To simplify the presentation we have restricted our language to a pure pi calculus of names and channel types, and omitted recursive types. In Section 7 we discuss the changes necessary to remove these restrictions; some of these extensions are required by the examples in Section 2. We have also omitted the original pi calculus choice $(+)$ and name-matching constructs, which have little interaction with the type system.

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*. A type variable X can occur in its dual form $\overline{X}$. Duality for all session types is defined recursively by the equations in Figure 2. The type variables X_i, in both normal and dual forms, are bound in $\&\langle\, l_i(T_i \leqslant X_i \leqslant T'_i) \,:\, S_i\,\rangle_{1\leqslant i\leqslant n}$ and $\oplus\langle\, l_i(T_i \leqslant X_i \leqslant T'_i) \,:\, S_i\,\rangle_{1\leqslant i\leqslant n}$ and are free otherwise. We require each label in a branch or choice type to carry exactly one type variable. This is for notational convenience and could easily be generalized. We identify types up to α-equivalence and assume when necessary that the names of bound type variables are different from each other and from the names of free type variables. Substitution of types for type variables in types is defined recursively in Figure 3.

The syntax of processes is defined by the grammar in Figure 4. 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. 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 non-standard. Binding occurrences of names are x in $(\nu x{:}T)P$ and y in $x^p?[y{:}T].P$. In $(\nu x{:}T)P$, both x^+ and x^- may occur in P, and both are bound. In $x^p?[y{:}T].P$, only y (unpolarized) may occur in P. This will become clear when the type system is presented, in Section 3.4. 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.

Most of the syntax of processes is standard. $\mathbf{0}$ is the inactive process, $|$ is parallel composition and $(\nu x{:}T)P$ declares a local channel x with two ports, x^+ and x^-, of types T and $\overline{T}$ respectively, for use in P. The process $x^p?[y{:}T].P$ receives the name y, which has type T, on port x^p, and then executes P. The process $x^p![y^q].P$ outputs the name y^q on port x^p and then executes P. The process $x^p \triangleright \{l_i(T_i \leqslant X_i \leqslant T'_i) : P_i\}_{1\leqslant i\leqslant n}$ offers a choice of subsequent behaviours on port x^p. One of the P_i can be selected as the continuation process by sending on port $x^{\overline{p}}$ the appropriate label, l_i, and an accompanying type T

Session types	S	$::=$	X	*type variable*
		$\mid$	$\overline{X}$	*dual of type variable*
		$\mid$	end	*terminated session*
		$\mid$	$?[T].S$	*input*
		$\mid$	$![T].S$	*output*
		$\mid$	$\&\langle\, l_i(T_i \leqslant X_i \leqslant T'_i) : S_i \,\rangle_{1\leqslant i\leqslant n}$	*branch*
		$\mid$	$\oplus\langle\, l_i(T_i \leqslant X_i \leqslant T'_i) : S_i \,\rangle_{1\leqslant i\leqslant n}$	*choice*
Types	T	$::=$	S	*session type*
		$\mid$	$\ulcorner[T]$	*standard channel type*
		$\mid$	Top	*top type*
		$\mid$	Bot	*bottom type*

Fig. 1. Types

$$
\begin{array}{rcl}
\overline{X} & = & \overline{X} \\
\overline{\overline{X}} & = & X \\
\overline{\mathsf{end}} & = & \mathsf{end} \\
\overline{?[T].S} & = & ![T].\overline{S} \\
\overline{![T].S} & = & ?[T].\overline{S} \\
\overline{\&\langle\, l_i(T_i \leqslant X_i \leqslant T'_i) : S_i \,\rangle_{1\leqslant i\leqslant n}} & = & \oplus\langle\, l_i(T_i \leqslant X_i \leqslant T'_i) : \overline{S_i} \,\rangle_{1\leqslant i\leqslant n} \\
\overline{\oplus\langle\, l_i(T_i \leqslant X_i \leqslant T'_i) : S_i \,\rangle_{1\leqslant i\leqslant n}} & = & \&\langle\, l_i(T_i \leqslant X_i \leqslant T'_i) : \overline{S_i} \,\rangle_{1\leqslant i\leqslant n} \\
\overline{\mathsf{Top}} & = & \mathsf{Bot} \\
\overline{\mathsf{Bot}} & = & \mathsf{Top}
\end{array}
$$

Fig. 2. Duality

$$
\begin{array}{rcl}
X\{U/X\} & = & U \\
\overline{X}\{U/X\} & = & \overline{U} \\
Y\{U/X\} & = & Y \quad \text{if } Y \neq X \\
\overline{Y}\{U/X\} & = & \overline{Y} \quad \text{if } Y \neq X \\
\mathsf{end}\{U/X\} & = & \mathsf{end} \\
(?[T].S)\{U/X\} & = & ?[T\{U/X\}].S\{U/X\} \\
(![T].S)\{U/X\} & = & ![T\{U/X\}].S\{U/X\} \\
(\&\langle\, l_i(T_i \leqslant X_i \leqslant U_i) : S_i \,\rangle_{1\leqslant i\leqslant n})\{U/X\} & = & \\
 & & \&\langle\, l_i(T_i\{U/X\} \leqslant X_i \leqslant U_i\{U/X\}) : S_i\{U/X\} \,\rangle_{1\leqslant i\leqslant n}{}^{*} \\
(\oplus\langle\, l_i(T_i \leqslant X_i \leqslant U_i) : S_i \,\rangle_{1\leqslant i\leqslant n})\{U/X\} & = & \\
 & & \oplus\langle\, l_i(T_i\{U/X\} \leqslant X_i \leqslant U_i\{U/X\}) : S_i\{U/X\} \,\rangle_{1\leqslant i\leqslant n}{}^{*} \\
\ulcorner[T]\{U/X\} & = & \ulcorner[T\{U/X\}] \\
\mathsf{Top}\{U/X\} & = & \mathsf{Top} \\
\mathsf{Bot}\{U/X\} & = & \mathsf{Bot}
\end{array}
$$

*where $X \notin \{X_1, \ldots, X_n\}$

Fig. 3. Substitution of types for type variables in types

$$
\begin{array}{rcll}
P, Q & ::= & \mathbf{0} & \textit{terminated process} \\
 & \mid & P \mid Q & \textit{parallel combination} \\
 & \mid & !P & \textit{replication} \\
 & \mid & x^p?[y{:}T].P & \textit{input} \\
 & \mid & x^p![y^p].P & \textit{output} \\
 & \mid & (\nu x{:}T)P & \textit{channel creation} \\
 & \mid & x^p \triangleright \{l_i(T_i \leqslant X_i \leqslant T_i) : P_i\}_{1\leqslant i\leqslant n} & \textit{branch} \\
 & \mid & x^p \triangleleft l(T).P & \textit{choice}
\end{array}
$$

Fig. 4. Processes

$$
\begin{array}{rcl}
x^q\{u^p/v\} & = & x^q \quad \text{if } x \neq v \\
x\{u^p/v\} & = & u_i^{p_i} \quad \text{if } x = v \\
\mathbf{0}\{u^p/v\} & = & \mathbf{0} \\
(P \mid Q)\{u^p/v\} & = & P\{u^p/v\} \mid Q\{u^p/v\} \\
(!P)\{u^p/v\} & = & !(P\{u^p/v\}) \\
(x^q?[y{:}T].P)\{u^p/v\} & = & x^q\{u^p/v\}?[y{:}T].P\{u^p/v\} \\
(x^q![y^r].P)\{u^p/v\} & = & x^q\{u^p/v\}![y^r\{u^p/v\}].P\{u^p/v\} \\
((\nu x{:}T)P)\{u^p/v\} & = & (\nu x{:}T)P\{u^p/v\} \\
(x^q \triangleright \{l_i(T_i \leqslant X_i \leqslant U_i) : P_i\}_{1\leqslant i\leqslant n}\{u^p/v\} & = & \\
\multicolumn{3}{r}{x^q\{u^p/v\} \triangleright \{l_i(T_i \leqslant X_i \leqslant U_i) : P_i\{u^p/v\}\}_{1\leqslant i\leqslant n}} \\
(x^q \triangleleft l(T).P)\{u^p/v\} & = & x^q\{u^p/v\} \triangleleft l(T).P\{u^p/v\}
\end{array}
$$

Fig. 5. Substitution of names for names in processes

$$
\begin{array}{rcl}
\mathbf{0}\{U/X\} & = & \mathbf{0} \\
(P \mid Q)\{U/X\} & = & P\{U/X\} \mid Q\{U/X\} \\
(x^q?[y{:}T].P)\{U/X\} & = & x^q?[y{:}T\{U/X\}].P\{U/X\} \\
(!P)\{U/X\} & = & !(P\{U/X\}) \\
(x^q![y^r].P)\{U/X\} & = & x^q![y^r].P\{U/X\} \\
((\nu x{:}T)P)\{U/X\} & = & (\nu x{:}T\{U/X\})P\{U/X\} \\
(x^q \triangleright \{l_i(T_i \leqslant X_i \leqslant U_i) : P_i\}_{1\leqslant i\leqslant n}\{U/X\} & = & \\
\multicolumn{3}{r}{x^q \triangleright \{l_i(T_i\{U/X\} \leqslant X_i \leqslant U_i\{U/X\}) : P_i\{U/X\}\}_{1\leqslant i\leqslant n}{}^*} \\
(x^q \triangleleft l(T).P)\{U/X\} & = & x^q \triangleleft l(T\{U/X\}).P\{U/X\}
\end{array}
$$

*where $X \notin \{X_1, \ldots, X_n\}$

Fig. 6. Substitution of types for type variables in processes

such that $T_i \leqslant T \leqslant T'_i$, as explained in Section 2. The process $x^p \triangleleft l(T).P$ sends the label l and type T on port x^p in order to make a selection from an offered range of options, and then executes P. We sometimes write $(\nu\widetilde{x}{:}\widetilde{T})P$ as shorthand for $(\nu x_1{:}T_1)\ldots(\nu x_n{:}T_n)P$.

Substitution of polarized names for unpolarized names, $P\{x^{\widetilde{p}}/y\}$, is defined recursively in Figure 5. Substitution of types for type variables in processes, $P\{T/X\}$, is defined recursively in Figure 6. In both cases we assume that α-conversion is used if necessary to avoid variable capture.

Example: a multithreaded server

To illustrate our language further, we use a running example of a server which spawns a separate thread to handle each request from a client. We assume that the types int and real are available, along with elementary operations on them.

The server provides the services sqr and plus, described by the session type

$$S \;=\; \&\langle\, \mathsf{sqr}(\mathsf{int} \leqslant X \leqslant \mathsf{real}) : ?[X].![X].\mathsf{end},\;\; \mathsf{plus}(\mathsf{int} \leqslant X \leqslant \mathsf{real}) : ?[X].?[X].![X].\mathsf{end}\,\rangle.$$

An individual run of the protocol is executed by the process thread, defined by

$$\begin{aligned}\mathsf{thread}(x) \;&=\; x \triangleright \{\; \mathsf{sqr}(\mathsf{int} \leqslant X \leqslant \mathsf{real}) : x?[y{:}X].x![y^2].\mathbf{0},\\ &\qquad\quad \mathsf{plus}(\mathsf{int} \leqslant X \leqslant \mathsf{real}) : x?[y{:}X].x?[z{:}X].x![y+z].\mathbf{0}\;\}.\end{aligned}$$

The natural definition of the server, which receives a session channel on u, passes it to a new thread, and is ready to receive another session channel, is

$$\mathsf{server}(u) \;=\; u?[x{:}S].(\mathsf{thread}(x) \mid \mathsf{server}(u)).$$

In the absence of recursive process definitions, we use standard pi calculus programming techniques to convert this definition into a replicated process in which a message on the “trigger” channel t causes a copy of the replicated part to be made:

$$\mathsf{repserver}(t,u) \;=\; !(t?[].u?[x{:}S].(\mathsf{thread}(x) \mid t![].\mathbf{0})).$$

Here the message on t has no content and is a pure synchronization. To conform strictly to our syntax, we could send a dummy value on t, but we will not bother.

The top-level server consists of repserver in parallel with a message on t to extract the first copy of the replicated part:

$$\mathsf{server}(u) \;=\; (\nu t{:}\,\widehat{}\,[])(\mathsf{repserver}(t,u) \mid t![].\mathbf{0}).$$

A possible client is defined by

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

and a complete system is

$$\mathsf{system}(u) \;=\; \mathsf{client}(u) \mid \mathsf{server}(u).$$

$$
\begin{array}{rcll}
P \mid \mathbf{0} & \equiv & P & \text{SC-Unit} \\
P \mid Q & \equiv & Q \mid P & \text{SC-Comm} \\
P \mid (Q \mid R) & \equiv & (P \mid Q) \mid R & \text{SC-Assoc} \\
!P & \equiv & P \mid !P & \text{SC-Rep} \\
(\nu x{:}T)P \mid Q & \equiv & (\nu x{:}T)(P \mid Q) \text{ if } x, x^+, x^- \notin fn(Q) & \text{SC-Extr} \\
(\nu x{:}\widehat{\ }[T])\mathbf{0} & \equiv & \mathbf{0} & \text{SC-Nil} \\
(\nu x{:}\mathsf{end})\mathbf{0} & \equiv & \mathbf{0} & \text{SC-NilS} \\
(\nu x{:}T)(\nu y{:}U)P & \equiv & (\nu y{:}U)(\nu x{:}T)P & \text{SC-Switch}
\end{array}
$$

Fig. 7. Structural congruence

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

$$\frac{1 \leqslant j \leqslant n}{x^p \triangleright \{l_i(T_i \leqslant X_i \leqslant U_i) : P_i\}_{1\leqslant i\leqslant n} \mid x^{\bar{p}} \triangleleft l_j(T).Q \xrightarrow{x,l_j(T)} P_j\{T/X_j\} \mid Q}\ \text{R-Select}$$

$$\frac{P \xrightarrow{\alpha,l(U)} P' \quad \alpha \neq x}{(\nu x{:}T)P \xrightarrow{\alpha,l(U)} (\nu x{:}T)P'}\ \text{R-New} \qquad \frac{P \xrightarrow{x,l(U)} P'}{(\nu x{:}T)P \xrightarrow{\tau,\_} (\nu x{:}tail(T,l(U)))P'}\ \text{R-NewS}$$

$$\frac{P \xrightarrow{\alpha,l(T)} P'}{P \mid Q \xrightarrow{\alpha,l(T)} P' \mid Q}\ \text{R-Par} \qquad \frac{P' \equiv P \quad P \xrightarrow{\alpha,l(T)} Q \quad Q \equiv Q'}{P' \xrightarrow{\alpha,l(T)} Q'}\ \text{R-Cong}$$

Fig. 8. The reduction relation

3.2. *Operational semantics*

Following one of the standard approaches to pi calculus semantics (Milner 1991) we define an operational semantics by means of 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 is closed under the equations in Figure 7. The structural congruence rules are standard. Rule SC-Nil 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 8. To enable our Type Preservation Theorem to be stated (Theorem 1, Section 4), reductions are annotated with labels of the form $\alpha, l(T)$. These labels indicate the channel name and branch selection label, if any, which are involved in each reduction. Consider a reduction which

$$
\begin{array}{rcl}
tail(?[T].S, \_) & = & S \\
tail(![T].S, \_) & = & S \\
tail(\&\langle\, l_i(T_i \leqslant X_i \leqslant U_i) : S_i \,\rangle_{1\leqslant i\leqslant n}, l_j(T)) & = & S_j\{T/X_j\} \\
tail(\oplus\langle\, l_i(T_i \leqslant X_i \leqslant U_i) : S_i \,\rangle_{1\leqslant i\leqslant n}, l_j(T)) & = & S_j\{T/X_j\}
\end{array}
$$

Fig. 9. The *tail* function

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 transmission of a choice label then l is that label and T is the associated type, otherwise $l(T) = \_$. We assume that the label $\_$ does not occur as a choice label. The labels have no semantic significance and would be omitted in any implementation.

Rule R-COM is the standard communication reduction for the pi calculus. Substitution of a polarized name for and unpolarized name is defined in Figure 5. The channel on which communication takes place, and the name which is transmitted, are polarized, perhaps as the result of substitutions arising from earlier reductions.

Rule R-SELECT resolves a choice between labelled processes, and instantiates polymorphism, by sending a label and a type 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 9. Rules R-PAR and R-CONG are standard.

Example: execution of the multithreaded server

Continuing the running example, we show a reduction sequence including complete execution of the client. Each step is either a reduction, a conversion by structural congruence, or an application of the definition of a process, and in each case we indicate which rule justifies it. For clarity we show the channel involved in the reduction even when it should be converted to τ by a surrounding ν.

First we follow the execution of $(\nu x{:}S)(\mathsf{clientbody}(x^-) \mid \mathsf{thread}(x^+))$, omitting the top-level $(\nu x{:}S)$.

$$\begin{array}{rcll}
\mathsf{clientbody}(x^-) & \mid & \mathsf{thread}(x^+) & \\
 & = & & \text{definition} \\
x^- \triangleleft \mathsf{sqr(int)}.x^-![3].x^-?[y{:}\mathsf{int}].\mathbf{0} & \mid & & \\
x^+ \triangleright \{\ \mathsf{sqr}(\mathsf{int} \leqslant X \leqslant \mathsf{real}) : \ldots, & & \mathsf{plus}(\mathsf{int} \leqslant X \leqslant \mathsf{real}) : \ldots\ \} & \\
 & \downarrow & x, \mathsf{sqr(int)} & \text{R-SELECT} \\
x^-![3].x^-?[y{:}\mathsf{int}].\mathbf{0} & \mid & x?[y{:}\mathsf{int}].x![y^2].\mathbf{0} & \\
 & \downarrow & x & \text{R-COM} \\
x^-?[y{:}\mathsf{int}].\mathbf{0} & \mid & x![9].\mathbf{0} & \\
 & \downarrow & x & \text{R-COM} \\
\mathbf{0} & \mid & \mathbf{0} &
\end{array}$$

We now show how a copy of repserver is extracted from server, and how client sends a session channel to repserver. This is not specific to session types, and simply illustrates recursive behaviour in pi calculus.

$$
\begin{array}{rcl}
 & \mathsf{system}(u) & \\
 & = & \text{definition}\\
\mathsf{client}(u) & \mid & \mathsf{server}(u)\\
 & = & \text{definition}\\
(\nu x{:}S)u![x^+].\mathsf{clientbody}(x^-) & \mid & (\nu t{:}\widehat{[\,]})(\mathsf{repserver}(t,u) \mid t![\,].\mathbf{0})\\
 & \equiv & \text{definition}\\
(\nu x{:}S)u![x^+].\mathsf{clientbody}(x^-) & \mid & \\
\multicolumn{3}{r}{(\nu t{:}\widehat{[\,]})(!(t?[\,].u?[z{:}S].(\mathsf{thread}(z) \mid t![\,].\mathbf{0})) \mid t![\,].\mathbf{0})}\\
 & \equiv & \textsc{SC-Rep}\\
(\nu x{:}S)u![x^+].\mathsf{clientbody}(x^-) & \mid & \\
\multicolumn{3}{r}{(\nu t{:}\widehat{[\,]})(t?[\,].u?[z{:}S].(\mathsf{thread}(z) \mid t![\,].\mathbf{0}) | !(t?[\,].u?[z{:}S].(\mathsf{thread}(z) \mid t![\,].\mathbf{0})) \mid t![\,].\mathbf{0})}\\
 & \downarrow & t \quad \textsc{R-Com}\\
(\nu x{:}S)u![x^+].\mathsf{clientbody}(x^-) & \mid & \\
\multicolumn{3}{r}{(\nu t{:}\widehat{[\,]})(u?[z{:}S].(\mathsf{thread}(z) \mid t![\,].\mathbf{0}) | !(t?[\,].u?[z{:}S].(\mathsf{thread}(z) \mid t![\,].\mathbf{0})) \mid \mathbf{0})}\\
 & \equiv & \textsc{SC-Unit}\\
(\nu x{:}S)u![x^+].\mathsf{clientbody}(x^-) & \mid & \\
\multicolumn{3}{r}{(\nu t{:}\widehat{[\,]})(u?[z{:}S].(\mathsf{thread}(z) \mid t![\,].\mathbf{0}) | !(t?[\,].u?[z{:}S].(\mathsf{thread}(z) \mid t![\,].\mathbf{0})))}\\
 & = & \text{definition}\\
(\nu x{:}S)u![x^+].\mathsf{clientbody}(x^-) & \mid & \\
\multicolumn{3}{r}{(\nu t{:}\widehat{[\,]})(u?[z{:}S].(\mathsf{thread}(z) \mid t![\,].\mathbf{0}) \mid \mathsf{repserver}(t,u))}\\
 & \equiv & \textsc{SC-Extr}\\
(\nu x{:}S)(\nu t{:}\widehat{[\,]})(u![x^+].\mathsf{clientbody}(x^-) & \mid & u?[z{:}S].(\mathsf{thread}(z) \mid t![\,].\mathbf{0}) \mid \mathsf{repserver}(t,u))\\
 & \downarrow & u \quad \textsc{R-Com}\\
(\nu x{:}S)(\nu t{:}\widehat{[\,]})(\mathsf{clientbody}(x^-) & \mid & \mathsf{thread}(x^+) \mid t![\,].\mathbf{0} \mid \mathsf{repserver}(t,u))\\
 & \equiv & \textsc{SC-Extr}\\
(\nu x{:}S)(\mathsf{clientbody}(x^-) & \mid & \mathsf{thread}(x^+)) \mid (\nu t{:}\widehat{[\,]})(t![\,].\mathbf{0} \mid \mathsf{repserver}(t,u))\\
 & = & \text{definition}\\
(\nu x{:}S)(\mathsf{clientbody}(x^-) & \mid & \mathsf{thread}(x^+)) \mid \mathsf{server}(u)
\end{array}
$$

3.3. *Subtyping (Kernel $S_{\leqslant}$)*

The subtype relation is based on that of (Gay and Hole 1999, 2005), with extensions for bounded polymorphism. Because types contain type variables with upper and lower bounds, subtyping is defined relative to an environment $\Delta = T_1 \leqslant X_1 \leqslant T_1', \ldots, T_n \leqslant X_n \leqslant T_n'$ in which the order of bounded type variables is significant and any type variables in T_i, T_i' are taken from $\{X_1, \ldots, X_{i-1}\}$. The upper and lower bounds of the type variables in branch (&) and choice ($\oplus$) types do not change when moving up the subtype relation, analogously to Kernel $\mathrm{F}_{<:}$(Cardelli and Wegner 1985; Pierce 2002). Note that the Top and Bot types are extremal over all types, not just session types. We refer to this system of subtyping for session types as Kernel $S_{\leqslant}$ or just $S_{\leqslant}$. Our inductive definition of subtyping, according to the rules in Figure 10, is algorithmic in the terminology of (Pierce 2002) — that is to say, reflexivity and transitivity are theorems rather than definitions.

$$\vdash \emptyset \quad \textsc{E-Empty} \qquad\qquad \frac{\vdash \Delta \qquad \Delta \vdash T \leqslant U \qquad X \notin \Delta}{\vdash \Delta, T \leqslant X \leqslant U}\ \textsc{E-Extend}$$

$$\frac{\vdash \Delta}{\Delta \vdash T \leqslant \mathsf{Top}}\ \textsc{S-Top} \qquad\qquad \frac{\vdash \Delta}{\Delta \vdash \mathsf{Bot} \leqslant T}\ \textsc{S-Bot}$$

$$\frac{\vdash \Delta}{\Delta \vdash X \leqslant X}\ \textsc{S-Refl}_1 \qquad\qquad \frac{\vdash \Delta}{\Delta \vdash \overline{X} \leqslant \overline{X}}\ \textsc{S-Refl}_2$$

$$\frac{\vdash \Delta \quad T \leqslant X \leqslant U \in \Delta \quad \Delta \vdash U \leqslant V}{\Delta \vdash X \leqslant V}\ \textsc{S-Tr}_1 \qquad \frac{\vdash \Delta \quad T \leqslant X \leqslant U \in \Delta \quad \Delta \vdash V \leqslant T}{\Delta \vdash V \leqslant X}\ \textsc{S-Tr}_2$$

$$\frac{\vdash \Delta \quad T \leqslant X \leqslant U \in \Delta \quad \Delta \vdash \overline{T} \leqslant V}{\Delta \vdash \overline{X} \leqslant V}\ \textsc{S-Tr}_3 \qquad \frac{\vdash \Delta \quad T \leqslant X \leqslant U \in \Delta \quad \Delta \vdash V \leqslant \overline{U}}{\Delta \vdash V \leqslant \overline{X}}\ \textsc{S-Tr}_4$$

$$\frac{\vdash \Delta}{\Delta \vdash \mathsf{end} \leqslant \mathsf{end}}\ \textsc{S-End} \qquad\qquad \frac{\vdash \Delta}{\Delta \vdash {}^\wedge[T] \leqslant {}^\wedge[T]}\ \textsc{S-Chan}$$

$$\frac{\vdash \Delta \quad \Delta \vdash V \leqslant W \quad \Delta \vdash T \leqslant U}{\Delta \vdash ?[T].V \leqslant ?[U].W}\ \textsc{S-In} \qquad \frac{\vdash \Delta \quad \Delta \vdash V \leqslant W \quad \Delta \vdash U \leqslant T}{\Delta \vdash ![T].V \leqslant ![U].W}\ \textsc{S-Out}$$

$$\frac{m \leqslant n \quad \forall_{1 \leqslant i \leqslant m}(\vdash \Delta, T_i \leqslant X_i \leqslant U_i \text{ and } \Delta, T_i \leqslant X_i \leqslant U_i \vdash R_i \leqslant S_i)}{\Delta \vdash \&\langle\, l_i(T_i \leqslant X_i \leqslant U_i) : R_i \,\rangle_{1 \leqslant i \leqslant m} \leqslant \&\langle\, l_i(T_i \leqslant X_i \leqslant U_i) : S_i \,\rangle_{1 \leqslant i \leqslant n}}\ \textsc{S-Branch}$$

$$\frac{m \leqslant n \quad \forall_{1 \leqslant i \leqslant m}(\vdash \Delta, T_i \leqslant X_i \leqslant U_i \text{ and } \Delta, T_i \leqslant X_i \leqslant U_i \vdash R_i \leqslant S_i)}{\Delta \vdash \oplus\langle\, l_i(T_i \leqslant X_i \leqslant U_i) : R_i \,\rangle_{1 \leqslant i \leqslant n} \leqslant \oplus\langle\, l_i(T_i \leqslant X_i \leqslant U_i) : S_i \,\rangle_{1 \leqslant i \leqslant m}}\ \textsc{S-Choice}$$

Fig. 10. Subtyping rules

Here is an example of a subtyping derivation.

$$\frac{(*) \qquad \dfrac{\dfrac{(*) \qquad \Delta \vdash \mathsf{real} \leqslant \mathsf{real}}{\Delta \vdash X \leqslant \mathsf{real}}\ \textsc{S-Tr}_1 \qquad \dfrac{(*)}{\Delta \vdash \mathsf{end} \leqslant \mathsf{end}}\ \textsc{S-End}}{\Delta \vdash ?[X].\mathsf{end} \leqslant ?[\mathsf{real}].\mathsf{end}}\ \textsc{S-In}}{\emptyset \vdash \&\langle\, \mathsf{a}(\mathsf{int} \leqslant X \leqslant \mathsf{real}) : ?[X].\mathsf{end} \,\rangle \leqslant \&\langle\, \mathsf{a}(\mathsf{int} \leqslant X \leqslant \mathsf{real}) : ?[\mathsf{real}].\mathsf{end} \,\rangle}\ \textsc{S-Branch}$$

The environment Δ is $\mathsf{int} \leqslant X \leqslant \mathsf{real}$. Each hypothesis $(*)$ is $\vdash \mathsf{int} \leqslant \mathsf{real}$, which follows from the assumption $\mathsf{int} \leqslant \mathsf{real}$ by rule E-Extend.

3.4. *Type system*

The rules in Figure 11 inductively define judgements of the form $\Delta\,;\Gamma \vdash P$ where Δ is a list of type variables with upper and lower bounds, exactly as in Section 3.3, and Γ is an *environment*. Such a judgement means that the process P uses channels as specified by the types and bounds in $\Delta\,;\Gamma$. A process is either correctly typed or not; we do not assign types to processes. Implicitly, whenever a Δ-environment appears in a typing rule, the judgement $\vdash \Delta$ is one of the hypotheses.

Definition 1. An environment Γ is a function from optionally polarized names to types. The types Bot and Top are not allowed in an environment. If $x^p \in \mathit{dom}(\Gamma)$ 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 $x^p \notin \mathit{dom}(\Gamma)$ 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 conditions below.

For any environment Γ and any name x, the following conditions must hold.

1 If $x \in \mathit{dom}(\Gamma)$ then $x^+ \notin \mathit{dom}(\Gamma)$ and $x^- \notin \mathit{dom}(\Gamma)$.
2 If $x^p \in \mathit{dom}(\Gamma)$ and p is either $^+$ or $^-$ then $x \notin \mathit{dom}(\Gamma)$ and $x^{\overline{p}} \notin \mathit{dom}(\Gamma)$ and $\Gamma(x^p)$ is a session type.

Definition 2. 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 $S' = \overline{S}$.

Definition 3. Addition of a typed name to an environment is defined by

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

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

The typing rules are similar to those of (Gay and Hole 2005) and share two characteristic features which are worth pointing out here. The first is that as a typing derivation is constructed, the session types in the environment change to reflect the sequence of communication. This can be seen in rules T-InS, T-OutS, T-Offer and T-Choose. For example, consider the following typing derivation:

$$\frac{x^+ : \mathsf{end} \vdash \mathbf{0}}{x^+ : ![\mathsf{int}].\mathsf{end} \vdash x^+![2].\mathbf{0}}\ \textsc{T-OutS}$$

As a result, the Type Preservation Theorem (Theorem 1, Section 4) must, as usual for session types, 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

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

$$\frac{\Delta\,;\Gamma, x{:}\widehat{\ }[T] \vdash P}{\Delta\,;\Gamma \vdash (\nu x{:}\widehat{\ }[T])P}\,\textsc{T-New} \qquad \frac{\Delta\,;\Gamma, x^{+}{:}S, x^{-}{:}\overline{S} \vdash P}{\Delta\,;\Gamma \vdash (\nu x{:}S)P}\,\textsc{T-NewS}$$

$$\frac{\Delta \vdash T \leqslant ?[U].S \qquad \Delta\,;\Gamma, x^p{:}S, y{:}U \vdash P}{\Delta\,;\Gamma, x^p{:}T \vdash x^p?[y{:}U].P}\,\textsc{T-InS}$$

$$\frac{\Delta \vdash T \leqslant ![U].S \qquad \Delta\,;\Gamma, x^p{:}S \vdash P}{\Delta\,;(\Gamma, x^p{:}T) + y^q{:}U \vdash x^p![y^q].P}\,\textsc{T-OutS}$$

$$\frac{\Delta \vdash T \leqslant \widehat{\ }[V] \qquad \Delta \vdash V \leqslant U \qquad \Delta\,;\Gamma, x{:}T, y{:}U \vdash P}{\Delta\,;\Gamma, x{:}T \vdash x?[y{:}U].P}\,\textsc{T-In}$$

$$\frac{\Delta \vdash T \leqslant \widehat{\ }[V] \qquad \Delta \vdash U \leqslant V \qquad \Delta\,;\Gamma, x{:}T \vdash P}{\Delta\,;(\Gamma, x{:}T) + y^q{:}U \vdash x![y^q].P}\,\textsc{T-Out}$$

$$\frac{\Delta \vdash T \leqslant \&\langle\, l_i(T_i \leqslant X_i \leqslant U_i) : S_i \,\rangle_{1\leqslant i\leqslant n} \qquad \forall_{1\leqslant i\leqslant n}(\Delta, T_i \leqslant X_i \leqslant U_i; \Gamma, x^p{:}S_i \vdash P_i)}{\Delta\,;\Gamma, x^p{:}T \vdash x^p \triangleright \{l_i(T_i \leqslant X_i \leqslant U_i) : P_i\}_{1\leqslant i\leqslant n}}\,\textsc{T-Offer}$$

$$\frac{\begin{array}{ccc}\Delta \vdash T \leqslant \oplus\langle\, l_i(T_i \leqslant X_i \leqslant U_i) : S_i \,\rangle_{1\leqslant i\leqslant n} & & \Delta\,;\Gamma, x^p{:}S_j\{T/X_j\} \vdash P \\ l = l_j \in \{l_1, \ldots, l_n\} \qquad & \Delta \vdash T_j \leqslant U \leqslant U_j & \qquad U \neq \mathsf{Bot}, \mathsf{Top}\end{array}}{\Delta\,;\Gamma, x^p{:}T \vdash x^p \triangleleft l(U).P}\,\textsc{T-Choose}$$

Fig. 11. Typing rules

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 $\Gamma + y^q : \widetilde{U}$ requires that the name y^q does not occur in Γ and is therefore not used in the continuation process P after it has been sent on x^p. This use of addition is based on the linear type system of (Kobayashi et al. 1999).

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. This point is discussed further by (Gay and Hole 2005). In rule T-Offer each branch is typed with the corresponding type for the channel, including the appropriate upper and lower bounds for the type variable; in rule T-Choose the continuation process is typed with the appropriately-instantiated polymorphic channel type. A crucial feature of the system is that polymorphism cannot be instantiated with the types Bot or Top, as indicated by the condition $U \neq \mathsf{Bot}, \mathsf{Top}$ in rule T-Choose. This condition guarantees that if a type variable has at least one bound which is not Bot or Top then any type instantiating it has the same structure as that bound. Rule T-NewS creates both ends of a new channel with dual types. The use of S indicates that the type must be a session type, meaning either a definite (non-variable) session type or a type variable which can be proved to be bounded by a definite session type.

Example: typing the multithreaded server

We now show typing derivations for some of the processes in the running example. First, thread. The final step of the derivation is

$$\dfrac{\begin{array}{c}\emptyset \vdash S \leqslant \&\langle\, \mathsf{sqr}(\mathsf{int} \leqslant X \leqslant \mathsf{real}) : ?[X].![X].\mathsf{end}, \\ \qquad\qquad \mathsf{plus}(\mathsf{int} \leqslant X \leqslant \mathsf{real}) : ?[X].?[X].![X].\mathsf{end}\,\rangle \\ \mathsf{int} \leqslant X \leqslant \mathsf{real};\ x{:}?[X].![X].\mathsf{end} \vdash x?[y{:}X].x![y^2].\mathbf{0} \\ \mathsf{int} \leqslant X \leqslant \mathsf{real};\ x{:}?[X].?[X].![X].\mathsf{end} \vdash x?[y{:}X].x?[z{:}X].x![y+z].\mathbf{0}\end{array}}{\begin{array}{l}\emptyset;\ x{:}S \quad\vdash\quad x \triangleright \{\ \mathsf{sqr}(\mathsf{int} \leqslant X \leqslant \mathsf{real}) : x?[y{:}X].x![y^2].\mathbf{0}, \\ \qquad\qquad\qquad\quad \mathsf{plus}(\mathsf{int} \leqslant X \leqslant \mathsf{real}) : x?[y{:}X].x?[z{:}X].x![y+z].\mathbf{0}\ \}\end{array}}\ \textsc{T-Offer}$$

The subtyping judgement (first hypothesis) follows immediately from the definition of S. The derivation of the second hypothesis is as follows (the third is similar). We omit the subtyping judgements, which are simply instances of reflexivity. We also make use of the necessary adaptations for an expression language allowing y^2 to be typed.

$$\dfrac{\dfrac{\dfrac{\text{rules for expressions}}{\mathsf{int} \leqslant X \leqslant \mathsf{real};\ y{:}X \vdash y^2{:}X} \qquad \dfrac{x{:}\mathsf{end}, y{:}X \text{ completed}}{\mathsf{int} \leqslant X \leqslant \mathsf{real};\ x{:}\mathsf{end}, y{:}X \vdash \mathbf{0}}\ \textsc{T-Nil}}{\mathsf{int} \leqslant X \leqslant \mathsf{real};\ x{:}![X].\mathsf{end}, y{:}X \vdash x![y^2].\mathbf{0}}\ \textsc{T-OutS}}{\mathsf{int} \leqslant X \leqslant \mathsf{real};\ x{:}?[X].![X].\mathsf{end} \vdash x?[y{:}X].x![y^2].\mathbf{0}}\ \textsc{T-InS}$$

Next, client. Again we omit the subtyping judgements. We also omit the empty Δ-environments.

$$\dfrac{\dfrac{\dfrac{\emptyset \vdash \mathsf{int} \leqslant \mathsf{int} \leqslant \mathsf{real} \qquad \dfrac{u{:}{}^\frown[S] \vdash 3{:}\mathsf{int} \qquad \dfrac{\dfrac{u{:}{}^\frown[S], x^-{:}\mathsf{end}, y{:}\mathsf{int} \text{ completed}}{u{:}{}^\frown[S], x^-{:}\mathsf{end}, y{:}\mathsf{int} \vdash \mathbf{0}}\ \textsc{T-Nil}}{u{:}{}^\frown[S], x^-{:}?[\mathsf{int}].\mathsf{end} \vdash x^-?[y{:}\mathsf{int}].\mathbf{0}}\ \textsc{T-InS}}{u{:}{}^\frown[S], x^-{:}![\mathsf{int}].?[\mathsf{int}].\mathsf{end} \vdash x^-![3].x^-?[y{:}\mathsf{int}].\mathbf{0}}\ \textsc{T-OutS}}{u{:}{}^\frown[S], x^-{:}\overline{S} \vdash x^- \triangleleft \mathsf{sqr}(\mathsf{int}).x^-![3].x^-?[y{:}\mathsf{int}].\mathbf{0}}\ \textsc{T-Choose}}{u{:}{}^\frown[S], x^+{:}S, x^-{:}\overline{S} \vdash u![x^+].x^- \triangleleft \mathsf{sqr}(\mathsf{int}).x^-![3].x^-?[y{:}\mathsf{int}].\mathbf{0}}\ \textsc{T-Out}}{u{:}{}^\frown[S] \vdash (\nu x{:}S)u![x^+].x^- \triangleleft \mathsf{sqr}(\mathsf{int}).x^-![3].x^-?[y{:}\mathsf{int}].\mathbf{0}}\ \textsc{T-NewS}$$

Observe that in the second hypothesis of T-Choose, the type of x^- is the type of the sqr branch of $\overline{S}$ with int substituted for X. This derivation also shows how after $u![x^+]$, x^+ is not in the environment above T-Out.

In these example derivations we have omitted several subtyping judgements. For example, in the general form of rule T-OutS there is a hypothesis $\Delta \vdash T \leqslant ![U].S$. In our derivations for the running example, these hypotheses are always straightforward instances of reflexivity of subtyping, because T is always an explicit type expression of the correct form. In more complex examples, T could be a type variable X, and in this case it would be necessary to make use of information (from Δ) about the bounds of X in order to construct a derivation of $\Delta \vdash X \leqslant ![U].S$. This would be seen in a typing derivation for the example in Section 2.5.

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, and in particular is similar to the corresponding proof in (Gay and Hole 2005). By *communication error* we mean an attempt to transmit a message of a form that the receiver is not prepared for. There are several possibilities.

1 Sending a value whose type does not match (i.e. is not a subtype of) the type declared by the receiver. For example:

$$x^{+}?[y{:}\mathsf{int}].\mathbf{0} \mid x^{-}![2.5].\mathbf{0}$$

or

$$x^{+}?[y{:}?[\mathsf{int}].\mathsf{end}].\mathbf{0} \mid (\nu z{:}?[?[\mathsf{int}].\mathsf{end}].\mathsf{end})x^{-}![z^{+}].\mathbf{0}.$$

2 Selecting a label that is not one of the options offered by the receiver. For example:

$$x^{+} \rhd \{\ \mathsf{sqr}(\mathsf{int} \leqslant X \leqslant \mathsf{real}) : x?[a{:}X].x![a^2].\mathbf{0}\ \} \mid x^{-} \lhd \mathsf{cos}(\mathsf{real}).x^{-}![2].x^{-}?[y{:}\mathsf{real}].\mathbf{0}$$

3 Instantiating polymorphism with a type that is not within the declared bounds. For example:

$$x^{+} \rhd \{\ \mathsf{sqr}(\mathsf{int} \leqslant X \leqslant \mathsf{real}) : x?[a{:}X].x![a^2].\mathbf{0}\ \} \mid x^{-} \lhd \mathsf{sqr}(?[\mathsf{int}].\mathsf{end}).\mathbf{0}$$

4 Selecting a label when the receiver is expecting a message. For example:

$$x^{+}?[y{:}\mathsf{int}].\mathbf{0} \mid x^{-} \lhd \mathsf{sqr}(?[\mathsf{int}].\mathsf{end}).\mathbf{0}$$

Similarly, sending a message when the receiver is expecting a label.

5 Duplication of a channel port. For example:

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

In cases 2 and 4 the operational semantics does not define a reduction. We regard these cases as situations in which it appears superficially that a communication could be possible, because there is an input on x^{+} and an output on x^{-}, but communication does not take place because of a mismatch between the kind of input and the kind of output.

All of the above are also examples of *immediate* communication errors, meaning that the send and receive involved are the first action in each process. In general, a communication error could arise after a number of correct communication steps.

Our strategy for proving that a correctly-typed process has no communication errors is to prove that it has no immediate communication errors (Theorem 2) and that a successful communication step results in another correctly-typed process (Theorem 1).

We first prove the necessary lemmas showing that typings are preserved by structural congruence and by substitutions of names for names and types for type variables.

At the top level we are interested in the execution of *closed* processes which are typable in an empty environment: $\emptyset\,;\emptyset \vdash P$.

It is worth noting that because we do not assign types to terms, many of the complexities of $\mathrm{F}_{<:}$(meets and joins, minimal types and so on) do not arise in our system.

Lemma 1. $\overline{T\{U/X\}} = \overline{T}\{U/X\}$.

Proof. Straightforward induction on the structure of T. □

Lemma 2. $\mathit{tail}(\overline{S}, l(U)) = \overline{\mathit{tail}(S, l(U))}$.

Proof. Directly from the definition of *tail*, using Lemma 1. □

Lemma 3 (Weakening in subtyping judgements). If $\vdash \Delta$ and $\Delta \vdash T \leqslant U$ and $\vdash \Delta, \Delta'$ then $\Delta, \Delta' \vdash T \leqslant U$ and the derivations have the same size.

Proof. Straightforward induction on the derivation of $\Delta \vdash T \leqslant U$. □

Lemma 4 (Subtyping and duality). If $\Delta \vdash T \leqslant U$ then $\Delta \vdash \overline{U} \leqslant \overline{T}$ and the derivations have the same size.

Proof. Straightforward induction on the derivation of $\Delta \vdash T \leqslant U$. □

Lemma 5 (Reflexivity of subtyping). $\Delta \vdash T \leqslant T$.

Proof. Straightforward induction on the structure of T. □

Lemma 6 (Transitivity of subtyping). If $\Delta \vdash T \leqslant U$ and $\Delta \vdash U \leqslant V$ then $\Delta \vdash T \leqslant V$.

Proof. By induction on the sum of the sizes of the derivations of $\Delta \vdash T \leqslant U$ and $\Delta \vdash U \leqslant V$. If either derivation finishes with S-REFL$_i$ or S-END, or if the first derivation finishes with S-BOT, or if the second derivation finishes with S-TOP, then the conclusion is immediate. If the derivation of $\Delta \vdash T \leqslant U$ finishes with an application of S-TR$_i$ then one of the four following cases applies.

— S-TR$_1$: In this case $T = X$, and we have $T_1 \leqslant X \leqslant T_2 \in \Delta$ and $\Delta \vdash T_2 \leqslant U$. The induction hypothesis gives $\Delta \vdash T_2 \leqslant V$, and S-TR$_1$ gives $\Delta \vdash X \leqslant V$ as required.

— S-TR$_2$: In this case $U = X$, and we have $T_1 \leqslant X \leqslant T_2 \in \Delta$ and $\Delta \vdash T \leqslant T_1$. The last rule in the derivation of $\Delta \vdash X \leqslant V$ must be S-TR$_1$ (the cases of S-REFL$_1$ and S-TOP have been eliminated), and we have $\Delta \vdash T_2 \leqslant V$. The derivation of $\vdash \Delta$ contains a derivation of $\Delta' \vdash T_1 \leqslant T_2$, for some $\Delta' \subset \Delta$. By Lemma 3, $\Delta \vdash T_1 \leqslant T_2$ with a derivation of the same size. The induction hypothesis applied to $\Delta \vdash T \leqslant T_1$ and $\Delta \vdash T_1 \leqslant T_2$ gives $\Delta \vdash T \leqslant T_2$. Applying the induction hypothesis to this judgement and $\Delta \vdash T_2 \leqslant V$ gives $\Delta \vdash T \leqslant V$ as required.

— S-TR$_3$: In this case $T = \overline{X}$, and we have $T_1 \leqslant X \leqslant T_2 \in \Delta$ and $\Delta \vdash \overline{T_1} \leqslant U$. The induction hypothesis gives $\Delta \vdash \overline{T_1} \leqslant V$, and S-TR$_3$ gives $\Delta \vdash \overline{X} \leqslant V$ as required.

— S-TR$_4$: In this case $U = \overline{X}$, and we have $T_1 \leqslant X \leqslant T_2 \in \Delta$ and $\Delta \vdash T \leqslant \overline{T_2}$. The last rule in the derivation of $\Delta \vdash X \leqslant V$ must be S-TR$_3$ (the cases of S-REFL$_2$ and S-TOP have been eliminated), and we have $\Delta \vdash \overline{T_1} \leqslant V$. The derivation of $\vdash \Delta$ contains a derivation of $\Delta' \vdash T_1 \leqslant T_2$, for some $\Delta' \subset \Delta$. By Lemma 4, $\Delta' \vdash \overline{T_2} \leqslant \overline{T_1}$ with a derivation of the same size. By Lemma 3, $\Delta \vdash T_1 \leqslant T_2$, with a derivation again of the same size. The induction hypothesis applied to $\Delta \vdash T \leqslant \overline{T_2}$ and $\Delta \vdash \overline{T_2} \leqslant \overline{T_1}$ gives $\Delta \vdash T \leqslant \overline{T_1}$. Applying the induction hypothesis to this judgement and $\Delta \vdash \overline{T_1} \leqslant V$ gives $\Delta \vdash T \leqslant V$ as required.

If the derivation of $\Delta \vdash U \leqslant V$ finishes with an application of S-TR$_i$ then there are four cases which are similar to the cases above. The remaining possibilities are that the derivations of $\Delta \vdash T \leqslant U$ and $\Delta \vdash U \leqslant V$ finish with applications of the same rule, namely one of S-CHAN, S-IN, S-OUT, S-BRANCH and S-CHOICE. These cases proceed by straightforward use of the induction hypothesis. □

Lemma 7 (Substitution of types preserves subtyping).
If $\Delta, T_1 \leqslant X \leqslant T_2, \Delta' \vdash T \leqslant U$ and $\Delta \vdash T_1 \leqslant V$ and $\Delta \vdash V \leqslant T_2$ then $\Delta, \Delta'\{V/X\} \vdash T\{V/X\} \leqslant U\{V/X\}$.

Proof. Straightforward induction on the derivation of $\Delta, T_1 \leqslant X \leqslant T_2, \Delta' \vdash T \leqslant U$. □

Lemma 8. If $\Delta\,;\Gamma, x^p : S \vdash P$ and S is a session type and $x^p \notin \mathit{fn}(P)$ then $S = \mathsf{end}$.

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

Lemma 9. If $\Delta\,;\Gamma \vdash P$ and $x \notin \mathit{dom}(\Gamma)$ and T is not a session type then $\Delta\,;\Gamma, x{:}T \vdash P$.

Proof. A straightforward induction on the derivation of $\Delta\,;\Gamma \vdash P$, 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 10. If $\Delta\,;\Gamma \vdash P$ and $x^p \notin \mathit{dom}(\Gamma)$ then $\Delta\,;\Gamma, x^p{:}\mathsf{end} \vdash P$.

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

Lemma 11. If $\Delta\,;\Gamma, x^p{:}\mathsf{end} \vdash P$ and $x^p \notin \mathit{fn}(P)$ then $\Delta\,;\Gamma \vdash P$.

Proof. A straightforward induction on the derivation of $\Delta\,;\Gamma, x^p{:}\mathsf{end} \vdash P$. □

Lemma 12. If $\Delta\,;\Gamma \vdash P$ then $\mathit{fn}(P) \subseteq \mathit{dom}(\Gamma)$.

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

Lemma 13 (Structural congruence preserves typing).
If $\Delta\,;\Gamma \vdash P$ and $P \equiv Q$ then $\Delta\,;\Gamma \vdash Q$.

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 SC-EXTR, in both directions, in the case involving a session type.

(Left-to-right): We have

$$\dfrac{\dfrac{\Delta\,;\Gamma_1, x^+ : S, x^- : \overline{S} \vdash P}{\Delta\,;\Gamma_1 \vdash (\nu x : S)P}\ \textsc{T-NewS} \qquad \Delta\,;\Gamma_2 \vdash Q}{\Delta\,;\Gamma_1 + \Gamma_2 \vdash (\nu x : S)P \mid Q}\ \textsc{T-Par}$$

which can be rearranged to give

$$\dfrac{\dfrac{\Delta\,;\Gamma_1, x^+ : S, x^- : \overline{S} \vdash P \qquad \Delta\,;\Gamma_2 \vdash Q}{\Delta\,;(\Gamma_1 + \Gamma_2), x^+ : S, x^- : \overline{S} \vdash P \mid Q}\ \textsc{T-Par}}{\Delta\,;\Gamma_1 + \Gamma_2 \vdash (\nu x : S)(P \mid Q)}\ \textsc{T-NewS}$$

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

(Right-to-left): We have

$$\dfrac{\dfrac{\Delta\,;\Gamma_1 \vdash P \qquad \Delta\,;\Gamma_2 \vdash Q}{\Delta\,;\Gamma, x^+ : S, x^- : \overline{S} \vdash P \mid Q}\ \textsc{T-Par}}{\Delta\,;\Gamma \vdash (\nu x : S)(P \mid Q)}\ \textsc{T-NewS}$$

where $\Gamma_1 + \Gamma_2 = \Gamma, x^+ : S, x^- : \overline{S}$. By assumption, $x^+ \notin \mathit{fn}(Q)$ and $x^- \notin \mathit{fn}(Q)$. If $S \neq \mathsf{end}$ (hence also $\overline{S} \neq \mathsf{end}$) then Lemma 8 gives $x^+, x^- \notin \mathit{dom}(\Gamma_2)$. Therefore $\Gamma_1 = \Gamma_1', x^+ : S, x^- : \overline{S}$ with $\Gamma_1' + \Gamma_2 = \Gamma$. We can construct the derivation

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

If $S = \mathsf{end}$ (hence also $\overline{S} = \mathsf{end}$) then by Lemmas 10 and 11 we can remove x^+ and x^- (if they occur) from Γ_2 to give Γ_2', and add them to Γ_1 to give Γ_1', with the result that $\Delta\,;\Gamma_1' \vdash P$, $\Delta\,;\Gamma_2' \vdash Q$, $\Gamma_1' = \Gamma_1'', x^+ : \mathsf{end}, x^- : \mathsf{end}$ and $\Gamma_1'' + \Gamma_2' = \Gamma$. We can then construct the derivation

$$\dfrac{\dfrac{\Delta\,;\Gamma_1'', x^+ : \mathsf{end}, x^- : \mathsf{end} \vdash P}{\Delta\,;\Gamma_1'' \vdash (\nu x : \mathsf{end})P}\ \textsc{T-NewS} \qquad \Delta\,;\Gamma_2' \vdash Q}{\Delta\,;\Gamma \vdash (\nu x : \mathsf{end})P \mid Q}\ \textsc{T-Par}$$

□

Lemma 14 (Substitution of names preserves typing).
If $\Delta\,;\Gamma, x{:}U \vdash P$ and $\Delta \vdash T \leqslant U$ and $\Gamma + y^p{:}T$ is defined then $\Delta\,;\Gamma + y^p{:}T \vdash P\{y^p/x\}$.

Proof. By induction on the derivation of $\Delta\,;\Gamma, x{:}U \vdash P$. Note that T and U cannot be Bot or Top because of the assumptions on environments. The proof is similar to the proof of the substitution lemma given by (Gay and Hole 2005). Lemmas 6 and 12 are used. □

Lemma 15 (Substitution of types preserves typing). If $\Delta, T_1 \leqslant X \leqslant T_2, \Delta'\,;\Gamma \vdash P$ and $\Delta \vdash T_1 \leqslant T$ and $\Delta \vdash\leqslant T_2$ and $T \neq \mathsf{Bot}, \mathsf{Top}$ then $\Delta, \Delta'\{T/X\}\,;\Gamma\{T/X\} \vdash P\{T/X\}$.

Proof. By induction on the derivation of $\Delta, T_1 \leqslant X \leqslant T_2, \Delta' \,;\Gamma \vdash P$. $\square$

We now prove that reduction preserves typability. It is sufficient to prove this with an empty Δ-environment; the inductive cases do not require generalization of Δ.

Theorem 1 (Type preservation).

1 If $\emptyset\,;\Gamma \vdash P$ and $P \xrightarrow{\tau,\_} Q$ then $\emptyset\,;\Gamma \vdash Q$.
2 If $\emptyset\,;\Gamma, x^+{:}S, x^-{:}\overline{S} \vdash P$ and $P \xrightarrow{x,\_} Q$ then $\emptyset\,;\Gamma, x^+{:}\mathit{tail}(S,\_), x^-{:}\mathit{tail}(\overline{S},\_) \vdash Q$.
3 If $\emptyset\,;\Gamma, x^+{:}S, x^-{:}\overline{S} \vdash P$ and $P \xrightarrow{x,l(U)} Q$ then $\emptyset\,;\Gamma, x^+{:}\mathit{tail}(S,l(U)), x^-{:}\mathit{tail}(\overline{S},l(U)) \vdash Q$.
4 If $\emptyset\,;\Gamma, x{:}T \vdash P$ and $P \xrightarrow{x,\_} Q$ then $\emptyset\,;\Gamma, x{:}T \vdash Q$.

Proof. By induction on the derivation of the reduction, considering the cases appropriate for the form of the label. During the proof we omit the empty Δ-environment from the typing judgements.

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

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

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

$$\dfrac{\Gamma, x^+ : S, x^- : \overline{S} \vdash P}{\Gamma \vdash (\nu x : S)P}\ \text{T-NewS}$$

By the induction hypothesis (clause 2) we have

$$\Gamma, x^+ : \mathit{tail}(S, l(U), x^- : \mathit{tail}(\overline{S}, l(U)) \vdash P'$$

By Lemma 2, $\mathit{tail}(\overline{S}, l(U)) = \overline{\mathit{tail}(S, l(U))}$. So T-NewS gives

$$\Gamma \vdash (\nu x : \mathit{tail}(S, l(U)))P'$$

as required.
The cases for R-New, R-Par and R-Cong follow straightforwardly from the induction hypothesis, using Lemma 13 for R-Cong.

2 The important case is R-Com. We have

$$x^p?[y{:}T].P \mid x^{\bar{p}}![z^q].Q \xrightarrow{x,\_} P\{z^q/y\} \mid Q$$

and the form of the environment $\Gamma, x^+{:}S, x^-{:}\overline{S}$ means that p is either $^+$ or $^-$; without loss of generality assume $p = {}^+$. The derivation of $\Gamma, x^+{:}S, x^-{:}\overline{S} \vdash x^+?[y{:}T].P \mid x^-![z^q].Q$ ends as follows, where $\Gamma_1 + \Gamma_2 = \Gamma$ and $\Gamma_3 + z^q{:}V = \Gamma_2$.

$$\dfrac{\dfrac{\vdash S \leqslant ?[T].S_1 \quad \Gamma_1, x^+{:}S_1, y{:}T \vdash P}{\Gamma_1, x^+{:}S \vdash x^+?[y{:}T].P}\ \text{T-InS} \qquad \dfrac{\vdash \overline{S} \leqslant ![V].S_2 \quad \Gamma_3, x^-{:}S_2 \vdash Q}{\Gamma_2, x^-{:}\overline{S} \vdash x^-![z^q].Q}\ \text{T-OutS}}{\Gamma, x^+{:}S, x^-{:}\overline{S} \vdash x^+?[y{:}T].P \mid x^-![z^q].Q}\ \text{T-Par}$$

Because the Δ-environments are empty, S cannot be a type variable. Also, the general assumptions on environments mean that $S \neq \mathsf{Bot}$. Therefore the condition $S \leqslant ?[T].S_1$ means that $S = ?[U].S_3$ with $U \leqslant T$ and $S_3 \leqslant S_1$. So $\overline{S} = ![U].\overline{S_3}$ and the condition $\overline{S} \leqslant ![V].S_2$ means that $V \leqslant U$ and $\overline{S_3} \leqslant S_2$. By transitivity (Lemma 6) we have $V \leqslant T$, and Lemma 14 gives

$$(\Gamma_1, x^+{:}S_1) + z^q{:}V \vdash P\{z^q/y\}.$$

Using Lemma 14 again gives

$$\begin{gathered}(\Gamma_1, x^+{:}S_3) + z^q{:}V \vdash P\{z^q/y\}\\ \Gamma_3, x^-{:}\overline{S_3} \vdash Q.\end{gathered}$$

By using T-Par we can derive

$$(\Gamma_1, x^+{:}S_3) + z^q{:}\widetilde{V} + (\Gamma_3, x^-{:}\overline{S_3}) \vdash P\{z^q/y\} \mid Q$$

which is the desired judgement, because $\Gamma_1 + z^q{:}V + \Gamma_3 = \Gamma$ and $S_3 = \mathit{tail}(S, \_)$. The cases of R-New, R-Par and R-Cong again follow straightforwardly from the induction hypothesis.

3 The important case is R-Select. We have

$$x^p \triangleright \{l_i(T_i \leqslant X_i \leqslant U_i){:}P_i\}_{1\leqslant i\leqslant n} \mid x^{\bar{p}} \triangleleft l_j(T).Q \xrightarrow{x, l_j(T)} P_j\{T/X_j\} \mid Q$$

with $1 \leqslant j \leqslant n$. Again we assume without loss of generality that $p = {}^+$. The derivation of $\Gamma, x^+{:}S, x^-{:}\overline{S} \vdash x^p \triangleright \{l_i(T_i \leqslant X_i \leqslant U_i){:}P_i\}_{1\leqslant i\leqslant n} \mid x^{\bar{p}} \triangleleft l_j(T).Q$ ends as follows; $\Gamma_1 + \Gamma_2 = \Gamma$. Note that S must be of the form $\&\langle l_i(T_i \leqslant X_i \leqslant U_i) : S'_i\rangle_{1\leqslant i\leqslant m}$ with $m \leqslant n$ and, for each i, $S'_i \leqslant S_i$. Therefore $\overline{S} = \oplus\langle l_i(T_i \leqslant X_i \leqslant U_i) : \overline{S'_i}\rangle_{1\leqslant i\leqslant m}$, and T-Choose also requires that $\overline{S'_j} \leqslant S'$.

$$\dfrac{\dfrac{\forall_{1\leqslant i\leqslant n}(T_i \leqslant X_i \leqslant U_i; \Gamma_1, x^+{:}S_i \vdash P_i)}{\Gamma_1, x^+{:}S \vdash x^p \triangleright \{l_i(T_i \leqslant X_i \leqslant U_i){:}P_i\}_{1\leqslant i\leqslant n}} \qquad \dfrac{(*) \quad \Gamma_2, x^-{:}S'\{T/X_j\} \vdash Q}{\Gamma_2, x^-{:}\overline{S} \vdash x^{\bar{p}} \triangleleft l_j(T).Q}}{\Gamma, x^+{:}S, x^-{:}\overline{S} \vdash x^p \triangleright \{l_i(T_i \leqslant X_i \leqslant U_i){:}P_i\}_{1\leqslant i\leqslant n} \mid x^{\bar{p}} \triangleleft l_j(T).Q}$$

The additional hypotheses $(*)$ are $\vdash T_j \leqslant T \leqslant U_j$ and $T \neq \mathsf{Bot}, \mathsf{Top}$. By Lemma 15, $\Gamma_1, x^+{:}S_j\{T/X_j\} \vdash P_j\{T/X_j\}$. By Lemmas 7 and 14, $\Gamma_1, x^+{:}S'_j\{T/X_j\} \vdash P_j\{T/X_j\}$. Also by Lemma 14 we have $\Gamma_2, x^-{:}\overline{S'_j}\{T/X_j\} \vdash Q$. By using T-Par we can derive

$$\Gamma, x^+{:}S'_j\{T/X_j\}, x^-{:}\overline{S'_j}\{T/X_j\} \vdash P_j\{T/X_j\} \mid Q$$

which, because $\mathit{tail}(S, l_j(T)) = S'_j\{T/X_j\}$, is the required typing. The cases of R-New, R-Par and R-Cong again follow straightforwardly from the induction hypothesis.

4 The main case is R-Com, which follows by similar but simpler reasoning to that for clause (2). The others follow straightforwardly from the induction hypothesis. $\square$

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; the Type Preservation theorem guarantees that the property of being typable in a balanced environment is preserved by reduction.

In order to state a Runtime Safety theorem, we introduce some terminology.

Definition 4. A process is an x^p-*process* if it is ready to interact on x^p, i.e. if it has one of the following forms.

1 $x^p?[y{:}T].P$

2 $x^p![y^q].P$

3 $x^p \triangleright \{l_i(T_i \leqslant X_i \leqslant T_i) : P_i\}_{1\leqslant i\leqslant n}$

4 $x^p \triangleleft l(T).P$

Theorem 2 (Runtime safety). Let $\emptyset\,;\Gamma \vdash P$ where Γ is balanced.

1 If $P \equiv (\nu\widetilde{u}{:}\widetilde{T})(P_1 \mid \ldots \mid P_n)$ then, for any x, at most one of the P_i is an x^+-process and at most one is an x^--process.

2 If $P \equiv (\nu\widetilde{u}{:}\widetilde{T})(P_1 \,|\, P_2 \,|\, Q)$ and P_1 is an x^+-process and P_2 is an x^--process then either

(a) P_1 is an input and P_2 is an output (or vice versa), or

(b) P_1 is an offer and P_2 is a choose (or vice versa).

3 If $P \equiv (\nu\widetilde{u}{:}\widetilde{T})(x^p?[y{:}V].P_1 \mid x^{\overline{p}}![z^q].P_2 \mid Q)$ then either

(a) p is blank and among $\Gamma, \widetilde{u}{:}\widetilde{T}$ we have $x{:}\widehat{\ }[U]$ and $z^q{:}W$ with $\emptyset \vdash W \leqslant U \leqslant V$

(b) p is not blank and among $\Gamma, \widetilde{u}{:}\widetilde{T}$ we have $x^p{:}?[U].S$ and $x^{\overline{p}}{:}![U].\overline{S}$ and $z^q{:}W$ with $\emptyset \vdash W \leqslant U \leqslant V$.

4 If $P \equiv (\nu\widetilde{u}{:}\widetilde{T})(x^p \triangleright \{\, l_i(T_i \leqslant X_i \leqslant U_i) : P_i \,\}_{1\leqslant i\leqslant n} \,|\, x^{\overline{p}} \triangleleft l(U).Q \,|\, R)$ then $l \in \{l_1, \ldots, l_n\}$, $l = l_j$ say, and among $\Gamma, \widetilde{u}{:}\widetilde{T}$ we have $x^p{:}\&\langle\, l_i(T_i \leqslant X_i \leqslant U_i) : S_i \,\rangle_{1\leqslant i\leqslant m}$ with $m \leqslant n$ and $\emptyset \vdash T_j \leqslant U \leqslant U_j$.

Proof. By analyzing the final steps in the derivation of $\emptyset\,;\Gamma \vdash P$, and using the information in the hypotheses of the typing rules used. Note that the $\widetilde{T}$ may be session types or other types. □

At the top level a process P is typechecked in the empty environment. The Type Preservation theorem guarantees that as P reduces, each subsequent process Q is typable; the Runtime Safety theorem guarantees that Q has no immediate communication errors. Therefore no communication errors occur during the execution of P.

5. Typechecking

Several steps are necessary in order to convert the typing and subtyping rules of Section 3 into a practical typechecking algorithm. The first step is that the typing rules must be converted from declarative to algorithmic form; specifically, so that the splitting of the environment in T-Par can be calculated. This process is standard for type systems with linear features, and is described in detail by (Gay and Hole 2005) for the pi calculus with session types but without bounded polymorphism; the presence of bounded polymorphism and the Δ environment does not require any new techniques.

The second step is that it may be desirable to infer polarities, so that they can be eliminated from the top-level syntax. This can also be done by the technique described by (Gay and Hole 2005). The only penalty, as discussed in that paper, is that some deadlocked processes become untypable; this does not seem to be a significant loss.

The third step is to establish that the subtyping rules form an effective algorithm for checking instances of the subtype relation. The rules have already been presented in such a way that they have a natural interpretation as an algorithm; we now turn our attention to proving that this algorithm terminates.

5.1. *Decidability of subtyping*

The proof in this section closely follows the proof of decidability of subtyping for Kernel $F_{<:}$, as presented by (Pierce 2002)[Chapter 28].

Definition 5. The function $\mathit{weight}_\Delta(T)$ is defined recursively by the following clauses. The total size of the types in Δ plus the size of T decreases with each recursive call, so termination is guaranteed.

$$\begin{array}{lcl}
\mathit{weight}_{\Delta_1,T\leqslant X\leqslant U,\Delta_2}(X) & = & 1+\mathit{weight}_{\Delta_1}(T)+\mathit{weight}_{\Delta_1}(U)\\
\mathit{weight}_{\Delta_1,T\leqslant X\leqslant U,\Delta_2}(\overline{X}) & = & 1+\mathit{weight}_{\Delta_1}(T)+\mathit{weight}_{\Delta_1}(U)\\
\mathit{weight}_\Delta(\mathsf{Top}) & = & 1\\
\mathit{weight}_\Delta(\mathsf{Bot}) & = & 1\\
\mathit{weight}_\Delta(\mathsf{end}) & = & 1\\
\mathit{weight}_\Delta(\urcorner[T]) & = & 1+\mathit{weight}_\Delta(T)\\
\mathit{weight}_\Delta(?[T].S) & = & 1+\mathit{weight}_\Delta(S)+\mathit{weight}_\Delta(T)\\
\mathit{weight}_\Delta(![T].S) & = & 1+\mathit{weight}_\Delta(S)+\mathit{weight}_\Delta(T)\\
\mathit{weight}_\Delta(\&\langle\, l_i(T_i\leqslant X_i\leqslant U_i) : S_i\,\rangle_{1\leqslant i\leqslant n}) & = & 1+\Sigma_i\mathit{weight}_\Delta(T_i)+\Sigma_i\mathit{weight}_\Delta(U_i)\\
 & & +\Sigma_i\mathit{weight}_{\Delta,T_i\leqslant X_i\leqslant U_i}(S_i)\\
\mathit{weight}_\Delta(\oplus\langle\, l_i(T_i\leqslant X_i\leqslant U_i) : S_i\,\rangle_{1\leqslant i\leqslant n}) & = & 1+\Sigma_i\mathit{weight}_\Delta(T_i)+\Sigma_i\mathit{weight}_\Delta(U_i)\\
 & & +\Sigma_i\mathit{weight}_{\Delta,T_i\leqslant X_i\leqslant U_i}(S_i)
\end{array}$$

The function $\mathit{weight}(\Delta)$ is defined recursively by

$$\begin{array}{lcl}
\mathit{weight}(\emptyset) & = & 1\\
\mathit{weight}(\Delta,T\leqslant X\leqslant U) & = & 1+\mathit{weight}(\Delta)+\mathit{weight}_\Delta(T)+\mathit{weight}_\Delta(U)
\end{array}$$

The *weight* of a subtyping judgement is defined by

$$\mathit{weight}(\Delta\vdash T\leqslant U) \quad = \quad \mathit{weight}(\Delta)+\mathit{weight}_\Delta(T)+\mathit{weight}_\Delta(U).$$

The *weight* of a judgement that an environment is well-formed is defined by

$$\mathit{weight}(\vdash\Delta) \quad = \quad \mathit{weight}(\Delta).$$

Lemma 16. If $\vdash\Delta,\Delta'$ then $\mathit{weight}_{\Delta,\Delta'}(T)=\mathit{weight}_\Delta(T)$.

Proof. By induction on the structure of T. □

Theorem 3. The subtype relation is decidable.

Proof. The rules in Figure 10 define an algorithm for checking instances of the subtype relation. The rules are applied in a pattern-matching style, in the order in which they appear in the Figure. The condition that environments are well-formed ($\vdash\Delta$) does not need to be checked repeatedly, but it is checked for the top-level call (i.e. checking $\Delta\vdash$

$$\frac{m \leqslant n \qquad \forall_{1 \leqslant i \leqslant m}(\Delta \vdash V_i \leqslant T_i \leqslant U_i \leqslant W_i \text{ and } \Delta, T_i \leqslant X_i \leqslant U_i \vdash R_i \leqslant S_i)}{\Delta \vdash \&\langle\, l_i(T_i \leqslant X_i \leqslant U_i) : R_i \,\rangle_{1 \leqslant i \leqslant m} \leqslant \&\langle\, l_i(V_i \leqslant X_i \leqslant W_i) : S_i \,\rangle_{1 \leqslant i \leqslant n}} \textsc{S-Branch}$$

$$\frac{m \leqslant n \qquad \forall_{1 \leqslant i \leqslant m}(\Delta \vdash V_i \leqslant T_i \leqslant U_i \leqslant W_i \text{ and } \Delta, T_i \leqslant X_i \leqslant U_i \vdash R_i \leqslant S_i)}{\Delta \vdash \oplus\langle\, l_i(V_i \leqslant X_i \leqslant W_i) : R_i \,\rangle_{1 \leqslant i \leqslant n} \leqslant \oplus\langle\, l_i(T_i \leqslant X_i \leqslant U_i) : S_i \,\rangle_{1 \leqslant i \leqslant m}} \textsc{S-Choice}$$

Fig. 12. Modified subtyping rules for Full $S_{\leqslant}$

$T \leqslant U$ requires checking $\vdash \Delta$) and when the environment is enlarged in rules S-Branch and S-Choice. The algorithm terminates on all inputs because in each rule, the weight of each hypothesis (recursive call) is less than the weight of the conclusion (goal). The proof for rules S-Tr$_i$ makes use of Lemma 16. The proof for rule S-Branch (and similarly S-Choice) depends on the fact that $\mathit{weight}_{\Delta}(\&\langle\, l_i(T_i \leqslant X_i \leqslant U_i) : S_i \,\rangle_{1 \leqslant i \leqslant n})$ includes $\Sigma_i \mathit{weight}_{\Delta, T_i \leqslant X_i \leqslant U_i}(S_i)$. □

6. The generalization to Full $S_{\leqslant}$

We can consider a generalization of the subtype relation, as indicated in Section 2, in which the upper and lower bounds of type variables are allowed to vary. This requires modification of the subtyping rules S-Branch and S-Choice as shown in Figure 12. We call the resulting system Full $S_{\leqslant}$.

We can rerun the proofs of Section 4, leading to Type Preservation and Type Safety theorems for $S_{\leqslant}$. The only significantly different proof is transitivity of subtyping.

Lemmas 3–5 are true in Full $S_{\leqslant}$ and are proved exactly as before.

Definition 6. For a type T, $\mathit{size}(T)$ is defined to be the total number of nodes in the syntax tree of T, using clauses such as $\mathit{size}(?[T].S) = 1 + \mathit{size}(S) + \mathit{size}(T)$.

Lemma 17 (Transitivity of subtyping and narrowing in Full $S_{\leqslant}$). For all $n \geqslant 0$:

1 If $\mathit{size}(U) \leqslant n$ and $\Delta \vdash T \leqslant U$ and $\Delta \vdash U \leqslant V$ then $\Delta \vdash T \leqslant V$.

2 If $\mathit{size}(T) + \mathit{size}(U) \leqslant n$ and $\Delta, T \leqslant X \leqslant U, \Delta' \vdash V \leqslant W$ and $\Delta \vdash T \leqslant T'$ and $\Delta \vdash U' \leqslant U$ then $\Delta, T' \leqslant X \leqslant U', \Delta' \vdash V \leqslant W$.

Proof. This proof closely follows the corresponding proof for Full F$_{<:}$, as presented by (Pierce 2002)[Chapter 28]. The proof is by induction on n (we refer to this as the *outer* induction).

1 By induction (the *inner* induction) on the sum of the sizes of the derivations of $\Delta \vdash T \leqslant U$ and $\Delta \vdash U \leqslant V$. If either derivation finishes with S-Refl$_i$ or S-End, or if the first derivation finishes with S-Bot, or if the second derivation finished with S-Top, then the conclusion is immediate. If either derivation finishes with an application of S-Tr$_i$ then the argument is identical to the proof of Lemma 6. The remaining possibilities are that the derivations of $\Delta \vdash T \leqslant U$ and $\Delta \vdash U \leqslant V$ finish with applications of the same rule, namely one of S-Chan, S-In, S-Out, S-Branch and S-Choice. The first three of these proceed by straightforward use of the inner induction hypothesis. The last two cases are similar to each other; we consider S-Branch.

The types T, U, V have the form

$$\begin{array}{rcl} T & = & \&\langle\, l_i(T_i \leqslant X_i \leqslant T'_i) : T''_i\,\rangle_{1\leqslant i\leqslant m} \\ U & = & \&\langle\, l_i(U_i \leqslant X_i \leqslant U'_i) : U''_i\,\rangle_{1\leqslant i\leqslant n} \\ V & = & \&\langle\, l_i(V_i \leqslant X_i \leqslant V'_i) : V''_i\,\rangle_{1\leqslant i\leqslant r} \end{array}$$

with $m \leqslant n \leqslant r$ and, for each i,

$$\Delta \vdash V_i \leqslant U_i \leqslant T_i \leqslant T'_i \leqslant U'_i \leqslant V'_i \tag{4}$$

$$\Delta, T_i \leqslant X_i \leqslant T'_i \vdash T''_i \leqslant U''_i \tag{5}$$

$$\Delta, U_i \leqslant X_i \leqslant U'_i \vdash U''_i \leqslant V''_i \tag{6}$$

Applying the outer induction hypothesis (part 1) to (4) gives

$$\Delta \vdash V_i \leqslant T_i \leqslant T'_i \leqslant V'_i \tag{7}$$

Applying the outer induction hypothesis (part 2) to (6), using (4), gives

$$\Delta, T_i \leqslant X_i \leqslant T'_i \vdash U''_i \leqslant V''_i \tag{8}$$

Applying the outer induction hypothesis (part 1) to (5) and (8) gives

$$\Delta, T_i \leqslant X_i \leqslant T'_i \vdash T''_i \leqslant V''_i \tag{9}$$

Using (7) and (9) in an instance of S-Branch gives $\Delta \vdash T \leqslant V$.

2 By an inner induction on the size of the derivation of $\Delta, T \leqslant X \leqslant U, \Delta' \vdash V \leqslant W$, with a case analysis on the last rule. The interesting cases are the S-Tr$_i$ when X is the variable involved in the rule.

— S-Tr$_1$: In this case $V = X$ and we have $\Delta, T \leqslant X \leqslant U, \Delta' \vdash U \leqslant W$ with a smaller derivation. Applying the inner induction hypothesis gives

$$\Delta, T' \leqslant X \leqslant U', \Delta' \vdash U \leqslant W \tag{10}$$

Applying Lemma 3 to $\Delta \vdash U' \leqslant U$ gives

$$\Delta, T' \leqslant X \leqslant U', \Delta' \vdash U' \leqslant U \tag{11}$$

Applying the outer induction hypothesis (part 1) to (10) and (11) gives

$$\Delta, T' \leqslant X \leqslant U', \Delta' \vdash U' \leqslant W \tag{12}$$

and then S-Tr$_1$ gives $\Delta, T' \leqslant X \leqslant U', \Delta' \vdash X \leqslant W$ as required.

— S-Tr$_2$: This case is similar to S-Tr$_1$.

— S-Tr$_3$: In this case $V = \overline{X}$ and we have $\Delta, T \leqslant X \leqslant U, \Delta' \vdash \overline{T} \leqslant W$ with a smaller derivation. Applying the inner induction hypothesis gives

$$\Delta, T' \leqslant X \leqslant U', \Delta' \vdash \overline{T} \leqslant W \tag{13}$$

Applying Lemma 3 to $\Delta \vdash T \leqslant T'$ gives

$$\Delta, T' \leqslant X \leqslant U', \Delta' \vdash T \leqslant T' \tag{14}$$

Applying Lemma 4 to (14) gives

$$\Delta, T' \leqslant X \leqslant U', \Delta' \vdash \overline{T'} \leqslant \overline{T} \qquad (15)$$

Applying the outer induction hypothesis (part 1) to (13) and (15) gives

$$\Delta, T' \leqslant X \leqslant U', \Delta' \vdash \overline{T'} \leqslant W \qquad (16)$$

and then S-TR$_1$ gives $\Delta, T' \leqslant X \leqslant U', \Delta' \vdash \overline{X} \leqslant W$ as required.

— S-TR$_4$: This case is similar to S-TR$_3$. □

Lemmas 7–15 are true in Full $S_\leqslant$ and are proved exactly as before. The statement of the Type Preservation theorem is exactly the same as in Kernel $S_\leqslant$.

Theorem 4 (Type preservation in Full $S_\leqslant$).

1 If $\emptyset\,;\Gamma \vdash P$ and $P \xrightarrow{\tau,\_} Q$ then $\emptyset\,;\Gamma \vdash Q$.

2 If $\emptyset\,;\Gamma, x^+{:}S, x^-{:}\overline{S} \vdash P$ and $P \xrightarrow{x,\_} Q$ then $\emptyset\,;\Gamma, x^+{:}\mathit{tail}(S,\_), x^-{:}\mathit{tail}(\overline{S},\_) \vdash Q$.

3 If $\emptyset\,;\Gamma, x^+{:}S, x^-{:}\overline{S} \vdash P$ and $P \xrightarrow{x,l(U)} Q$ then $\emptyset\,;\Gamma, x^+{:}\mathit{tail}(S,l(U)), x^-{:}\mathit{tail}(\overline{S},l(U)) \vdash Q$.

4 If $\emptyset\,;\Gamma, x{:}T \vdash P$ and $P \xrightarrow{x,\_} Q$ then $\emptyset\,;\Gamma, x{:}T \vdash Q$.

Proof. Only the proof of part 3 is different from the proof of Theorem 1. The important case is R-SELECT. We have

$$x^p \triangleright \{l_i(T_i \leqslant X_i \leqslant U_i){:}P_i\}_{1\leqslant i\leqslant n} \mid x^{\bar{p}} \triangleleft l_j(T).Q \xrightarrow{x,l_j(T)} P_j\{T/X_j\} \mid Q$$

with $1 \leqslant j \leqslant n$. We assume without loss of generality that $p = {}^+$. The derivation of $\Gamma, x^+{:}S, x^-{:}\overline{S} \vdash x^p \triangleright \{l_i(T_i \leqslant X_i \leqslant U_i){:}P_i\}_{1\leqslant i\leqslant n} \mid x^{\bar{p}} \triangleleft l_j(T).Q$ ends as follows. Note that S must be of the form $\&\langle\, l_i(V_i \leqslant X_i \leqslant W_i) : S_i\,\rangle_{1\leqslant i\leqslant m}$ with $m \leqslant n$ and for each i, $\vdash T_i \leqslant V_i$ and $\vdash W_i \leqslant U_i$ and $\vdash S_i' \leqslant S_i$. Therefore $\overline{S} = \oplus\langle\, l_i(V_i \leqslant X_i \leqslant W_i) : \overline{S_i}\,\rangle_{1\leqslant i\leqslant m}$, and $\Gamma_1 + \Gamma_2 = \Gamma$. T-CHOOSE also requires that $\overline{S_j'} \leqslant S'$.

$$\dfrac{\dfrac{\forall_{1\leqslant i\leqslant n}(T_i \leqslant X_i \leqslant U_i;\, \Gamma, x^+{:}S_i \vdash P_i)}{\Gamma_1, x^+{:}S \vdash x^p \triangleright \{l_i(T_i \leqslant X_i \leqslant U_i){:}P_i\}_{1\leqslant i\leqslant n}} \qquad \dfrac{(*) \quad \Gamma_2, x^-{:}S'\{T/X_j\} \vdash Q}{\Gamma_2, x^-{:}\overline{S} \vdash x^{\bar{p}} \triangleleft l_j(T).Q}}{\Gamma, x^+{:}S, x^-{:}\overline{S} \vdash x^p \triangleright \{l_i(T_i \leqslant X_i \leqslant U_i){:}P_i\}_{1\leqslant i\leqslant n} \mid x^{\bar{p}} \triangleleft l_j(T).Q}$$

The additional hypotheses (*) are $\vdash V_j \leqslant T$, $\vdash T \leqslant W_j$ and $T \neq \mathsf{Bot}, \mathsf{Top}$. By transitivity, $\vdash T_j \leqslant T$ and $\vdash T \leqslant U_j$. The proof can now be completed exactly as in the proof of Theorem 1. □

The statement and proof of the Runtime Safety theorem are exactly the same as in Kernel $S_\leqslant$.

Theorem 5 (Runtime safety in Full $S_\leqslant$). Let $\emptyset\,;\Gamma \vdash P$ where Γ is balanced.

1 If $P \equiv (\nu\widetilde{u}{:}\widetilde{T})(P_1 \mid \ldots \mid P_n)$ then, for any x, at most one of the P_i is an x^+-process and at most one is an x^--process.

2 If $P \equiv (\nu\widetilde{u}{:}\widetilde{T})(P_1 \mid P_2 \mid Q)$ and P_1 is an x^+-process and P_2 is an x^--process then either

(a) P_1 is an input and P_2 is an output (or vice versa), or

(b) P_1 is an offer and P_2 is a choose (or vice versa).

3 If $P \equiv (\nu\widetilde{u}{:}\widetilde{T})(x^p?[y{:}V].P_1 \mid x^{\overline{p}}![z^q].P_2 \mid Q)$ then either

(a) p is blank and among $\Gamma, \widetilde{u}{:}\widetilde{T}$ we have $x{:}\widehat{\ }[U]$ and $z^q{:}W$ with $\emptyset \vdash W \leqslant U \leqslant V$

(b) p is not blank and among $\Gamma, \widetilde{u}{:}\widetilde{T}$ we have $x^p{:}?[U].S$ and $x^{\overline{p}}{:}![U].\overline{S}$ and $z^q{:}W$ with $\emptyset \vdash W \leqslant U \leqslant V$.

4 If $P \equiv (\nu\widetilde{u}{:}\widetilde{T})(x^p \triangleright \{\, l_i(T_i \leqslant X_i \leqslant U_i) : P_i \,\}_{1\leqslant i\leqslant n} \mid x^{\overline{p}} \triangleleft l(U).Q \mid R)$ then $l \in \{l_1, \ldots, l_n\}$, $l = l_j$ say, and among $\Gamma, \widetilde{u}{:}\widetilde{T}$ we have $x^p{:}\&\langle\, l_i(T_i \leqslant X_i \leqslant U_i) : S_i \,\rangle_{1\leqslant i\leqslant m}$ with $m \leqslant n$ and $\emptyset \vdash T_j \leqslant U \leqslant U_j$.

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

It turns out that subtyping in Full $S_{\leqslant}$ is undecidable, which can easily be shown by an encoding of subtyping problems from Full $\mathrm{F}_{<:}$. In the following definitions and results, "Full $\mathrm{F}_{<:}$" refers to the algorithmic presentation of the subtyping system in (Pierce 2002).

Definition 7. The function $[\![\cdot]\!]$ from types in Full $\mathrm{F}_{<:}$to Full $S_{\leqslant}$ is defined as follows.

$$\begin{array}{lcl}
[\![\mathsf{Top}]\!] & = & \mathsf{Top} \\
[\![X]\!] & = & X \\
[\![T \to U]\!] & = & ![[\![T]\!]].[\![U]\!] \\
[\![\forall X{<:}T.U]\!] & = & \oplus\langle\, l(X \leqslant [\![T]\!]) : [\![U]\!] \,\rangle
\end{array}$$

The function $[\![\cdot]\!]$ from subtyping environments in Full $\mathrm{F}_{<:}$to subtyping environments in Full $S_{\leqslant}$ is defined as follows.

$$\begin{array}{lcl}
[\![\emptyset]\!] & = & \emptyset \\
[\![\Gamma, X{<:}T]\!] & = & [\![\Gamma]\!], \mathsf{Bot} \leqslant X \leqslant [\![T]\!]
\end{array}$$

Proposition 1. For all types T, U in Full $\mathrm{F}_{<:}$, $\Gamma \vdash T{<:}U$ if and only if $[\![\Gamma]\!] \vdash [\![T]\!] \leqslant [\![U]\!]$.

Proof. Each direction is a straightforward induction on the derivation of the assumed judgement. The key to the proof is the match between the variance of the constructors. □

Theorem 6. Subtyping in Full $S_{\leqslant}$ is undecidable.

Proof. Subtyping in Full $\mathrm{F}_{<:}$is undecidable. □

It remains to be seen whether the undecidability of subtyping in Full $S_{\leqslant}$ is a serious practical problem. The subtyping rules can of course still be used as a semi-algorithm. It would be interesting to investigate whether restrictions on the form of types that are allowed as upper and lower bounds can restore decidability. Note, however, that the approach followed by (Katiyar and Sankar 1992) for completely bounded quantification is not sufficient. This is because branch and choice types have similar subtyping behaviour to record types, and record types have been shown to break the decidability of completely bounded quantification.

7. Conclusions and future work

Building on our previous theory of subtyping, we have proposed a system of bounded polymorphism for session types in the pi calculus. This allows polymorphic protocol specifications such that a client, when selecting a service, can consistently instantiate the types of all subsequent communications with the server. We have formalized the syntax, operational semantics and type system of an extended pi calculus, and proved that static typechecking guarantees absence of runtime communication errors. There are two versions of the system: Kernel $S_{\leqslant}$ and Full $S_{\leqslant}$; these differ in whether or not the bounds of type variables are allowed to vary when moving up the subtype relation. We have proved that subtyping is decidable in Kernel $S_{\leqslant}$ but undecidable in Full $S_{\leqslant}$.

There are some obvious possibilities for further development of $S_{\leqslant}$. The first is to allow recursive types, in order to describe protocols with repetitive behaviour. For example, modifying example (1) of Section 1 to

$$S = \&\langle\, \mathsf{go}\!:?[\mathsf{int}].![\mathsf{int}].S,\ \mathsf{quit}\!:\mathsf{end}\,\rangle$$

describes the protocol for a server that repeatedly offers the go behaviour; the quit option is now more meaningful. We included recursive types in our previous work on subtyping for session types (Gay and Hole 2005) and we would expect to proceed in a similar way.

The second is to investigate whether decidable subtyping can be achieved for a restricted, but still usefully flexible, version of Full $S_{\leqslant}$ in which the bounds of type variables can only take certain forms. For example, restricting bounds to be data types (not session types) might lead to decidability; another possibility might be to restrict the complexity, perhaps in terms of the depth of branch and choice types, of session types used as bounds.

Finally, the examples in Section 2 made use of basic data types such as int and real, implicitly assuming that the associated basic operations can be given types that make the examples work and do not cause problems for the typechecking algorithm. For very simple typings such as

$$\mathsf{int} \leqslant X \leqslant \mathsf{real}\,; a{:}X \vdash a^2{:}X$$

(Section 2.4) this is believable. For a more realistic expression language, perhaps approaching the complexity of $\mathrm{F}_{<:}$, it would be worth formalizing the complete system of pi calculus with expressions and checking that the necessary metatheoretic properties hold.

Acknowledgements

This research was begun in collaboration with the author's PhD student Malcolm Hole, who died on 28th February 2004; it was his suggestion to investigate bounded polymorphism as a way of increasing the expressive power of session types. He was funded by the EPSRC project "Novel Type Systems for Concurrent Programming Languages" (grants GR/L75177, GR/N39494). The author is grateful to the anonymous referees for their detailed reviews and helpful comments.

References

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

Carbone, M., Honda, K., Yoshida, N., Milner, R., Brown, G. and Ross-Talbot, S. (2006) A theoretical basis of communication-centred concurrent programming. W3C Web Services Choreography Working Group Report. `www.w3.org/2002/ws/chor/edcopies/theory/note.pdf`.

Cardelli, L. and Wegner, P. (1985) On understanding types, data abstraction, and polymorphism. *Computing Surveys* **17**(4):471–522.

Dezani-Ciancaglini, M., Mostrous, D., Yoshida, N. and Drossopolou, S. (2006) Session types for object-oriented languages. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, volume 4067 of *Lecture Notes in Computer Science*, pages 328–352. Springer.

Dezani-Ciancaglini, M., Yoshida, N., Ahern, A. and Drossopolou, S. (2005) A distributed object-oriented language with session types. In *Proceedings of the Symposium on Trustworthy Global Computing*, volume 3705 of *Lecture Notes in Computer Science*, pages 299–318. Springer.

Fähndrich, M., Aiken, M., Hawblitzel, C., Hodson, O., Hunt, G., Larus, J. R. and Levi, S. (2006) Language support for fast and reliable message-based communication in Singularity OS. In *EuroSys 2006*. ACM Press.

Garralda, P., Compagnoni, A. and Dezani-Ciancaglini, M. (2006) BASS: Boxed Ambients with Safe Sessions. In *Proceedings of the ACM Symposium on Principles and Practice of Declarative Programming*, pages 61–72. ACM Press.

Gay, S. J. (1993) A sort inference algorithm for the polyadic π-calculus. In *Proceedings of the 20th ACM Symposium on Principles of Programming Languages*. ACM Press.

Gay, S. J. and Hole, M. J. (1999) 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.

Gay, S. J. and Hole, M. J. (2005) Subtyping for session types in the pi calculus. *Acta Informatica* **42**(2/3):191–225.

Girard, J.-Y. (1972) *Interprétation fonctionelle et élimination des coupures dans l'arithmétique d'ordre supérieur*. Ph.D. thesis, University of Paris VII.

Girard, J.-Y. (1987) Linear Logic. *Theoretical Computer Science* **50**(1):1–102.

Honda, K. (1993) 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.

Honda, K., Vasconcelos, V. and Kubo, M. (1998) 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.

Katiyar, D. and Sankar, S. (1992) Completely bounded quantification is decidable. In *Proceedings of the ACM SIGPLAN Workshop on ML and its Applications*.

Kobayashi, N., Pierce, B. C. and Turner, D. N. (1999) Linearity and the Pi-Calculus. *ACM Transactions on Programming Languages and Systems* **21**(5):914–947.

Liu, X. and Walker, D. (1995) A polymorphic type system for the polyadic π-calculus. In *CONCUR'95: Proceedings of the International Conference on Concurrency Theory*, volume 962 of *Lecture Notes in Computer Science*. Springer.

Milner, R. (1991) The polyadic π-calculus: a tutorial. Technical Report 91-180, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh.

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

Neubauer, M. and Thiemann, P. (2004a) 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.

Neubauer, M. and Thiemann, P. (2004b) Session types for asynchronous communication. Manuscript.

Pierce, B. C. (2002) *Types and Programming Languages*. MIT Press.

Pierce, B. C. and Sangiorgi, D. (1993) Types and subtypes for mobile processes. In *Proceedings, Eighth Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press.

Pierce, B. C. and Sangiorgi, D. (1996) Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science* **6**(5):409–454.

Pierce, B. C. and Sangiorgi, D. (2000) Behavioral equivalence in the polymorphic pi-calculus. *Journal of the ACM* **47**(3).

Pierce, B. C. and Turner, D. N. (2000) Pict: A programming language based on the pi-calculus. In Plotkin, G., Stirling, C. and Tofte, M., editors, *Proof, Language and Interaction: Essays in Honour of Robin Milner*. MIT Press.

Reynolds, J. C. (1974) Towards a theory of type structure. In *Paris colloquium on programming*, volume 19 of *Lecture Notes in Computer Science*. Springer.

Sangiorgi, D. and Walker, D. (2001) *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press.

Takeuchi, K., Honda, K. and Kubo, M. (1994) 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.

Turner, D. N. (1996) *The Polymorphic Pi-Calculus: Theory and Implementation*. Ph.D. thesis, University of Edinburgh.

Vallecillo, A., Vasconcelos, V. T. and Ravara, A. (2006) Typing the behavior of software components using session types. *Fundamenta Informaticae* **73**(4):583–598.

Vasconcelos, V. T., Gay, S. J. and Ravara, A. (2006) Type checking a multithreaded functional language with session types. *Theoretical Computer Science* **368**(1–2):64–87.

Vasconcelos, V. T. and Honda, K. (1993) 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.

Vasconcelos, V. T., Ravara, A. and Gay, S. J. (2004) 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.

Yoshida, N. and Vasconcelos, V. T. (2006) Language primitives and type discipline for structured communication-based programming revisited - two systems for higher-order session communication. In *Procedings of the 1st International Workshop on Security and Rewriting Techniques (SecReT 2006)*, Electronic Notes in Theoretical Computer Science. Elsevier.