Bounded Polymorphism in Session Types

Malcolm Hole and Simon Gay

UNIVERSITY of GLASGOW

Department of Computing Science
University of Glasgow
Glasgow G12 8QQ
Scotland

TR-2003-132
March 2003

Bounded Polymorphism in Session Types

Simon Gay[1] and Malcolm Hole[2]

[1] Department of Computing Science, University of Glasgow, Glasgow G12 8QQ, UK.
Email: <simon@dcs.gla.ac.uk>
[2] Department of Computer Science, Royal Holloway, University of London,
Egham, Surrey TW20 0EX, UK. Email: <M.Hole@cs.rhul.ac.uk>

March 26, 2003

Abstract

We review work on session types and the π-calculus, and give an example identifying a use for bounded polymorphic types in this setting. We then define a variant of the π-calculus with the appropriate syntax, and propose a static bounded polymorphic type system, for which we prove type soundness. We use our rules to give a typing derivation and show the sequence of reductions in our example. Finally, we discuss related work and possible future work in this area.

This research was funded by the EPSRC project "Novel Type Systems for Concurrent Programming Languages" (GR/L75177,GR/N39494).

1 Introduction

Distributed client-server systems are 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 a client or server which is meant to follow a particular protocol, it is clearly 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 client-server 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. Session types were proposed (in the context of a language based on the π-calculus [13, 20]) by Takeuchi, Honda and Kubo [21] and have been studied further by Honda, Vasconcelos and Kubo [8] and the present authors [3, 4]. More recently Vallecillo, Vasconcelos and Ravara [23] have applied session types to the specification of component object systems, and Gay, Ravara and Vasconcelos [6] have begun to transfer session types from the π-calculus to a more conventional programming language.

In previous papers [3, 5] we increased the expressive power of session types in the π-calculus by defining a notion of subtyping, based on Pierce and Sangiorgi's [14, 15] system of subtyping in the π-calculus but extended to the additional type constructors involved in session types. The main application of subtyping is to allow certain kinds of server upgrades, which alter the protocol and hence its session type, to be made without removing the type-correctness of older clients which are unaware of the upgrade. However, the range of permissible upgrades is not as large as we would like, and the goal of the present paper is to extend it. We are still working within a language based on the π-calculus.

Our solution is to introduce a notion of bounded polymorphism, similar in general terms to kernel $\mathrm{F}_{<:}$[1, 16] but adapted to the π-calculus. As far as we know this is the first study of bounded polymorphism in the π-calculus, and the first study of any form of polymorphism in relation to session types. We discuss some previous work on polymorphism in the π-calculus in Section 7.

The rest of the paper is structured as follows. In Section 2 we review session types, illustrate their application to the specification of client-server protocols, describe the additional expressive power provided by subtyping, and explain how we can use bounded polymorphism to increase the expressive power still further. In Sections 3, 4 and 5 we formalise the syntax, type system and operational semantics of our version of the π-calculus, and outline the proof of soundness of the type system. In Section 6 we present example derivations, and in Section 7 we conclude, discuss related work, and indicate directions for further work.

2 Session Types in the Pi Calculus

Session types in the π-calculus provide the facility for a greater degree of structure in the interaction between a client and a server. Both parties participating in a sequence of communications 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 client and server, the whole dialogue taking place on a single channel.

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 occurence in a process. In this example, the server will use x^+ and the client x^-. x^+ and x^- each have an associated session type, which we will call S and $\overline{S}$, respectively.

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

The $\&\langle\ldots\rangle$ constructor (pronounced branch) in the type S 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). Conversely, the $\oplus\langle\ldots\rangle$ constructor (pronounced choice) in the type $\overline{S}$ indicates the making of a choice. Note that the pattern of sending and receiving for each label here is the opposite of that in the type of the server's port.

One possible implementation of a server and client using a channel with these types would be

$$\begin{array}{rcl} \mathsf{server} & = & x^+ \triangleright \{\mathsf{sin} : x^+ \mathbin{?} [a : \mathsf{int}] \,.\, x^+ \mathbin{!} [\mathsf{sin(a)}] \,.\, \mathbf{0}, \\ & & \quad \mathsf{sqr} : x^+ \mathbin{?} [a : \mathsf{int}] \,.\, x^+ \mathbin{!} [\mathsf{a}^2] \,.\, \mathbf{0}\} \\ \mathsf{client} & = & x^- \triangleleft \mathsf{sin} \,.\, x^- \mathbin{!} [90] \,.\, x^- \mathbin{?} [r : \mathsf{real}] \,.\, \mathbf{0} \end{array}$$

Here the server uses the $\triangleright\{\ldots\}$ construct (pronounced offer) to offer the labels sin and sqr. Each label then has a continuation process performing the necessary inputs and outputs as specified in the type. The client uses the $\triangleleft$ construct (pronounced choose) to choose the label sin from those on offer and then performs the appropriate sequence of outputs and inputs. As is usual in the π-calculus, we use parallel composition, $|$, to form a single process from two others and write typing judgements in the following way.

$$x^+ : S, x^- : \overline{S}, 90 : \mathsf{int} \vdash \mathsf{server} \mid \mathsf{client}$$

2.1 Subtyping

Subtyping allows us to upgrade a server whilst maintaining backward compatibility with clients that are written for the old version. Continuing our example, we might want to add a tangent operation to our server. We might also want to modify the server to accept real number inputs to the trigonometric operations. The type of the server's end of the new channel, S', would be

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

with the server now being implemented as

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

The client, unaware of the change made to the server, would still like to communicate with it on a channel where the server's end has type S. With our definition of subtyping this can happen, as it follows from the fact that $\mathsf{int} \leqslant \mathsf{real}$ and the fact that S' offers additional labels to those in S that $S \leqslant S'$.

2.2 Bounded Polymorphism

We can see that subtyping allows us to describe server upgrades in a structured way. There is a problem with this, however, in that there is a class of server upgrades for which subtyping alone is not sufficient. Consider now a client-server interaction involving a single addition operation. The communications again take place on a single channel, x, with two ports, x^+ and x^-. Here the type of x^+ is

$$S \quad = \quad \&\langle \mathsf{plus}\colon ?[\mathsf{real}]\,.\,?[\mathsf{real}]\,.\,![\mathsf{real}]\,.\,\mathsf{end}\rangle$$

A single label, plus, is offered, its continuation type showing that a real is input, followed by another, and finally the resulting real is output. The client side of the channel, x^-, has the complementary type

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

where a choice is made, albeit from the same single label. As one would expect, outputs exist in the continuation type where inputs are present in S and vice versa.

If we consider int to be a subtype of real, we might hope that we could use our 'plus server' to add two integers on channel x. A process with a server and client performing this interaction in parallel would look as follows.

$$\begin{array}{rl} P = & x^+ \triangleright \{\mathsf{plus}\colon x^+ \;?\; [a : \mathsf{real}] \,.\, x^+ \;?\; [b : \mathsf{real}] \,.\, x^+ \;!\; [a+b] \,.\, \mathbf{0}\} \\ | & x^- \triangleleft \mathsf{plus} \,.\, x^- \;!\; [7] \,.\, x^- \;!\; [11] \,.\, x^- \;?\; [r : \mathsf{int}] \,.\, \mathbf{0} \end{array}$$

Here, the client chooses the plus operation offered. It then sends the integer 7 followed by the integer 11 to the server, which adds these and sends back the result which the client binds to the name r. Unfortunately, however, the typing judgement

$$x^+ : S, x^- : \overline{S}, 7 : \mathsf{int}, 11 : \mathsf{int} \vdash P$$

is not correct, the reason being as follows: In our definition of subtyping, input is covariant and output is contravariant: $?[\mathsf{int}] \leqslant ?[\mathsf{real}]$ and $![\mathsf{real}] \leqslant ![\mathsf{int}]$. So, where the client wishes to output the integers to be added, it can use the port's capability to output real numbers (as $![\mathsf{real}] \leqslant ![\mathsf{int}]$) and the integers are promoted to reals inside the server, the result of adding them being typed as a real too. When receiving the result, however, the type system prevents the client from using the port's capability to input reals to input an integer (as $?[\mathsf{real}] \not\leqslant ?[\mathsf{int}]$) and thus the process is not typable.

One solution to this problem would be to have a separate server for each type. There are obvious disadvantages to this approach, though, and a more elegant solution would be preferable. We present a bounded polymorphic session type system with type variables and their upper bounds attached to labels in the offer construct. The process would now be written

$$\begin{array}{rl} Q = & x^+ \triangleright \{\mathsf{plus}(X \leqslant \mathsf{real})\colon x^+ \;?\; [a : X] \,.\, x^+ \;?\; [b : X] \,.\, x^+ \;!\; [a+b] \,.\, \mathbf{0}\} \\ | & x^- \triangleleft \mathsf{plus}(\mathsf{int}) \,.\, x^- \;!\; [7] \,.\, x^- \;!\; [11] \,.\, x^- \;?\; [r : \mathsf{int}] \,.\, \mathbf{0} \end{array}$$

where attached to the plus label in the server is a type variable X and an upper bound of real. The client can pass with the label any subtype of real, which is then substituted for

X in the continuation process. In this case, the client chooses plus with type int and our reduction rules allow this to reduce to the process

$$Q' = x^+ \mathbin{?} [a : \mathsf{int}] \,.\, x^+ \mathbin{?} [b : \mathsf{int}] \,.\, x^+ \mathbin{!} [a + b] \,.\, \mathbf{0} \mid x^- \mathbin{!} [7] \,.\, x^- \mathbin{!} [11] \,.\, x^- \mathbin{?} [r : \mathsf{int}] \,.\, \mathbf{0}$$

where both client and server are expecting to send and receive integers at every communication step. Our typing rules allow the following typing judgement to be correct.

$$x^+ : S', x^- : \overline{S'}, 7 : \mathsf{int}, 11 : \mathsf{int} \vdash Q$$

where

$$\begin{array}{rcl} S' & = & \&\langle \mathsf{plus}(X \leqslant \mathsf{real}) : ?[X] \,.\, ?[X] \,.\, ![X] \,.\, \mathsf{end}\rangle \\ \overline{S'} & = & \oplus\langle \mathsf{plus}(X \leqslant \mathsf{real}) : ![X] \,.\, ![X] \,.\, ?[X] \,.\, \mathsf{end}\rangle \end{array}$$

3 Syntax and Notation

Our language is based on a polyadic π-calculus with output prefixing [13] and is very similar syntactically to the language we proposed for session types with subtyping [3]. The inclusion of output prefixing is different from many recent presentations of the π-calculus, but it is essential because our type system must be able to impose an order on separate outputs on the same channel. We omit the original π-calculus choice construct, $P + Q$, and the matching construct which allows channel names to be tested for equality. We add a conditional process expression, however, written if b then P else Q where b is a boolean value, and therefore we also have a data type of booleans, as well as integers and reals. Other data types could be added along with appropriate primitive operations.

The most important addition to the syntax are the constructs for choosing between a collection of labelled processes while exchanging a type to be used in the continuation of an interaction, an extension of the constructs proposed by Honda et al. [8, 21]. As the aim of this paper is to look at bounded polymorphism in conjunction with session types, we do not include syntax for non-session channels. Neither do we include recursive processes or recursive types (although these could be added), and therefore omit the standard π-calculus replication operator, !. The need for linear control of session channels leads to the use of polarities on session ports which play a similar role to the polarities used on types by Kobayashi et al. [9, 10].

The type system has syntax for type variables and data types; these being the types that can be used as upper bounds. It also has constructors for session types similar to those proposed by Honda et al. [8, 21], but with type bindings attached to each label in a branch or offer type. Subtyping will be defined in Section 4.

In general we use lower case letters for channel names, upper case P, Q, R for processes, upper case T, U etc. for types, $l_1, \ldots, l_n$ for labels of choices, and $X_1, \ldots, X_n$ for type variables. We write $\tilde{x}$ for a finite sequence $x_1, \ldots, x_n$ of names, $\tilde{T}$ for a finite sequence $T_1, \ldots, T_n$ of types, and $\tilde{x} : \tilde{T}$ for a finite sequence $x_1 : T_1, \ldots, x_n : T_n$ of typed names.

3.1 Types

The syntax of types is defined by the following grammar.

$$
\begin{array}{llcl}
\text{Types} & T & ::= & B \mid S \\
\text{Type Bounds} & B & ::= & D \\
 & & \mid & X \quad \text{(type variable)} \\
\text{Data Types} & D & ::= & \mathsf{bool} \mid \mathsf{int} \mid \mathsf{real} \\
\text{Session Types} & S & ::= & \mathsf{end} \\
 & & \mid & ?[\tilde{T}] \,.\, S \\
 & & \mid & ![\tilde{T}] \,.\, S \\
 & & \mid & \&\langle l_i(X_i \leqslant B_i) : S_i \rangle_{i\in\{1,\ldots,n\}} \\
 & & \mid & \oplus\langle l_i(X_i \leqslant B_i) : S_i \rangle_{i\in\{1,\ldots,n\}}
\end{array}
$$

Free and bound type variables are defined as follows: variables $X_1, \ldots, X_n$ are bound in $\&\langle l_i(X_i \leqslant B_i) : S_i \rangle_{i\in\{1,\ldots,n\}}$ and $\oplus\langle l_i(X_i \leqslant B_i) : S_i \rangle_{i\in\{1,\ldots,n\}}$ and are free otherwise. We identify types up to α-equivalence.

The type S is a session type for one port of a session channel, say x^+. If x^+ has type S then x^- will have type $\overline{S}$, the dual (or complementary) type of S, which is defined inductively as follows.

$$
\begin{array}{c}
\overline{\mathsf{end}} = \mathsf{end} \\
\overline{?[\tilde{T}] \,.\, S} = ![\tilde{T}] \,.\, \overline{S} \qquad\qquad \overline{![\tilde{T}] \,.\, S} = ?[\tilde{T}] \,.\, \overline{S} \\
\overline{\&\langle l_i(X_i \leqslant B_i) : S_i \rangle_{i\in\{1,\ldots,n\}}} = \oplus\langle l_i(X_i \leqslant B_i) : \overline{S_i} \rangle_{i\in\{1,\ldots,n\}} \\
\overline{\oplus\langle l_i(X_i \leqslant B_i) : S_i \rangle_{i\in\{1,\ldots,n\}}} = \&\langle l_i(X_i \leqslant B_i) : \overline{S_i} \rangle_{i\in\{1,\ldots,n\}}
\end{array}
$$

As we have type variables, we define substitution, again inductively, as follows.

$$
\begin{array}{rcl}
D\{B/X\} & = & D \\
X\{B/X\} & = & B \\
Y\{B/X\} & = & X \quad \text{if } Y \neq X \\
\mathsf{end}\{B/X\} & = & \mathsf{end} \\
(?[\tilde{T}] \,.\, S)\{B/X\} & = & ?[\tilde{T}\{B/X\}] \,.\, S\{B/X\} \\
(![\tilde{T}] \,.\, S)\{B/X\} & = & ![\tilde{T}\{B/X\}] \,.\, S\{B/X\} \\
(\&\langle l_i(X_i \leqslant B_i) : S_i \rangle_{i\in\{1,\ldots,n\}})\{B/X\} & = & \&\langle l_i(X_i \leqslant B_i\{B/X\}) : S_i\{B/X\} \rangle_{i\in\{1,\ldots,n\}}{}^* \\
(\oplus\langle l_i(X_i \leqslant B_i) : S_i \rangle_{i\in\{1,\ldots,n\}})\{B/X\} & = & \oplus\langle l_i(X_i \leqslant B_i\{B/X\}) : S_i\{B/X\} \rangle_{i\in\{1,\ldots,n\}}{}^*
\end{array}
$$

*where $\forall i \in \{1, \ldots, n\}. X \neq X_i$

3.2 Processes

The syntax of processes is defined by the following grammar.

$$
\begin{array}{lcllcl}
P & ::= & \mathbf{0} & & & \\
 & \mid & P \mid Q & & \mid & x^p \triangleright \{l_i(X_i \leqslant B_i) : P_i\}_{i\in\{1,\ldots,n\}} \\
 & \mid & (\nu x^{\pm} : S)P & & \mid & x^p \triangleleft l(B) \,.\, P \\
 & \mid & x^p \,?\, [\tilde{y} : \tilde{T}] \,.\, P & & \mid & \mathsf{if}\ x\ \mathsf{then}\ P\ \mathsf{else}\ Q \\
 & \mid & x^p \,!\, [\tilde{y}] \,.\, P & & &
\end{array}
$$

Most of this syntax is fairly standard. $\mathbf{0}$ is the inactive process, $\mid$ is parallel composition and $(\nu x^{\pm} : S)P$ declares a local channel x with two ports, x^+ and x^-, of types S and $\overline{S}$

respectively for use in P. The process $x^p\,?\,[\tilde{y}:\tilde{T}]\,.\,P$ receives the names $\tilde{y}$, which have types $\tilde{T}$, on port x^p, and then executes P. The process $x^p\,!\,[\tilde{y}]\,.\,P$ outputs the names $\tilde{y}$ along the port x^p and then executes P. Process $x^p \triangleright \{l_i(X_i \leqslant B_i) : P_i\}_{i\in\{1,\ldots,n\}}$ 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 session port x^p the appropriate label, l_i, and an accompanying type B that is a subtype of B_i, as explained in Section 1. Process $x^p \triangleleft l(B)\,.\,P$ sends the label l and type B along port x^p in order to make a selection from an offered choice, and then executes P. The conditional expression, as one would expect, selects P or Q as the continuation process depending on the boolean value, b.

We define free and bound names as usual: x is bound in $(\nu x^\pm : S)P$, the names in $\tilde{y}$ are bound in $x^p\,?\,[\tilde{y}:\tilde{T}]\,.\,P$, and all other occurrences are free. Type variables $X_1,\ldots,X_n$ are bound in $x^p \triangleright \{l_i(X_i \leqslant B_i) : P_i\}_{i\in\{1,\ldots,n\}}$ and free in all other cases. A process with no free names or variables we call a program. We define α-equivalence as usual, and identify processes which are α-equivalent. We also define two substitution operations: $P\{\tilde{x}/\tilde{y}\}$ denotes P with the names $x_1,\ldots,x_n$ simultaneously substituted for $y_1,\ldots,y_n$; $P\{B/X\}$ denotes P with the type B substituted for X. In both cases, we assume that bound names/types are renamed if necessary to avoid capture of substituting names.

As usual we define a *structural congruence* relation, written $\equiv$, which helps to define the operational semantics. It is the smallest congruence (on α-equivalence classes of processes) closed under the following rules.

$$\begin{array}{rcll}
P \mid \mathbf{0} & \equiv & P & \textsc{S-Unit} \\
P \mid Q & \equiv & Q \mid P & \textsc{S-Comm} \\
P \mid (Q \mid R) & \equiv & (P \mid Q) \mid R & \textsc{S-Assoc} \\
(\nu x^p : T)P \mid Q & \equiv & (\nu x^p : T)(P \mid Q) \text{ if } x \text{ is not free in } Q & \textsc{S-Extr} \\
(\nu x^p : T)(\nu y^q : U)P & \equiv & (\nu y^q : U)(\nu x^p : T)P & \textsc{S-Switch} \\
x^p \triangleright \{l_i(X_i \leqslant B_i) : P_i\}_{i\in\{1,\ldots,n\}} & \equiv & x^p \triangleright \{l_{\sigma(i)}(X_{\sigma(i)} \leqslant B_{\sigma(i)}) : P_{\sigma(i)}\}_{i\in\{1,\ldots,n\}} & \\
 & & & \textsc{S-Offer}
\end{array}$$

In rule S-Offer, σ is a permutation on $\{1,\ldots,n\}$.

3.3 Environments

An environment is a sequence of type variables and their upper bounds followed by a set of typed names, written as follows.

$$X_1 \leqslant B_1,\ldots,X_m \leqslant B_m; x_1^{p_1} : T_1,\ldots,x_n^{p_n} : T_n$$

For an environment to be valid, each B_i can only refer to type variables X_1 to X_{i-1} and all names, $x_1^{p_1}$ to $x_n^{p_n}$, must be distinct.

We use $\Delta;\Gamma$, $\Delta_1;\Gamma_1$, $\Delta_2;\Gamma_2$ etc. to stand for environments, and if $\Delta = \emptyset$ we write just Γ. We write $x \in \Gamma$ to indicate that x is one of the names appearing in Γ. We then write $\Delta;\Gamma \vdash x^p : T$ to indicate that the type of x^p in $\Delta;\Gamma$ is T and $\Delta;\Gamma \vdash x^p \leqslant T$ to indicate that the type of x^p in $\Delta;\Gamma$ is a subtype of T. When $x^p \notin \Gamma$ we write $\Gamma, x^p : T$ for the set formed by adding $x^p : T$ to the set of typed names in Γ. When Γ_1 and Γ_2 have disjoint sets of names, we write Γ_1, Γ_2 for their union. Implicitly, *true* : bool and *false* : bool appear in every environment. We say that an environment is *completed* if it contains no session types except for end and *balanced* if, for every session port in the environment with some type S, the other port of that channel is also in the environment with type $\overline{S}$.

4 The Type System

4.1 Subtyping

The principles behind the definition of subtyping in our previous paper [3] have been described in Section 1. Our definition here is simplified due to the omission of non-session channel types and recursive types. Type variable bindings are added to the labels in the branch and choice rules, but are simply carried through. The main effect of introducing type variables is that subtyping judgements are now relative to some Δ, a finite sequence of type variables with upper bounds, equivalent to the first part of an environment.

Our definition of subtyping is algorithmic in Pierce's terminology [16] — that is to say, transitivity and reflexivity are theorems rather than definitions. We can also prove that if $T \leqslant U$ then $\overline{U} \leqslant \overline{T}$. Because we do not have recursive types, we have a simple inductive definition according to the rules in Figure 1.

$$\frac{}{\Delta \vdash D \leqslant D}\ \textsc{AS-Data} \qquad \frac{}{\Delta \vdash \mathsf{int} \leqslant \mathsf{real}}\ \textsc{AS-IntReal}$$

$$\frac{}{\Delta, X \leqslant B, \Delta' \vdash X \leqslant B}\ \textsc{AS-TVar} \qquad \frac{}{\Delta \vdash \mathsf{end} \leqslant \mathsf{end}}\ \textsc{AS-End}$$

$$\frac{\Delta \vdash V \leqslant W \quad \forall i \in \{1,\ldots,n\}.(\Delta \vdash T_i \leqslant U_i)}{\Delta \vdash ?[\tilde{T}]\,.\,V \leqslant ?[\tilde{U}]\,.\,W}\ \textsc{AS-In}$$

$$\frac{\Delta \vdash V \leqslant W \quad \forall i \in \{1,\ldots,n\}.(\Delta \vdash U_i \leqslant T_i)}{\Delta \vdash ![\tilde{T}]\,.\,V \leqslant ![\tilde{U}]\,.\,W}\ \textsc{AS-Out}$$

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

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

Figure 1: Subtyping Rules

4.2 Typing Rules

The typing rules are defined in Figure 2. T-Nil ensures that all interactions are finished when typing the nil process, **0**, by forcing the environment to be completed. In T-Par, the rule for parallel composition, note that the two environments must have identical sequences of type variables with upper bounds but distinct sets of names. This is because the upper bounds of free type variables are global assumptions, but a name representing a session port can only be used by one process at a time. The T-New rule forces the two ports of a session channel to have dual types when bound in order that sequences of communications on tha channel will not get out of step. T-Cond is standard.

The next four rules all involve communication on a session channel with the possibility that its type will be promoted according to the subtyping given in the hypothesis of the rule. Other than the subtyping, T-In is standard. In T-Out, the environment typing the process P does not include the $\tilde{y}$ to ensure that each y_i is only used by one process at a

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

$$\frac{\Delta;\Gamma, x^+ : S, x^- : \bar{S} \vdash P}{\Delta;\Gamma \vdash (\nu x^\pm : S)P}\ \textsc{T-New} \qquad \frac{\Delta;\Gamma \vdash P \quad \Delta;\Gamma \vdash Q \quad \Delta;\Gamma \vdash x \leqslant \mathsf{bool}}{\Delta;\Gamma \vdash \mathsf{if}\ x\ \mathsf{then}\ P\ \mathsf{else}\ Q}\ \textsc{T-Cond}$$

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

$$\frac{\Delta;\Gamma, x^p : S \vdash P \quad \Delta \vdash T \leqslant ![\tilde{U}]\,.\,S}{\Delta;\Gamma, x^p : T, \tilde{y} : \tilde{U} \vdash x^p\,!\,[\tilde{y}]\,.\,P}\ \textsc{T-Out}$$

$$\frac{\forall i \in \{1,\ldots,n\}.(\Delta, X_i \leqslant B_i;\Gamma, x^p : S_i \vdash P_i) \quad \Delta \vdash T \leqslant \&\langle l_i(X_i \leqslant B_i) : S_i\rangle_{i\in\{1,\ldots,n\}}}{\Delta;\Gamma, x^p : T \vdash x^p \triangleright \{l_i(X_i \leqslant B_i) : P_i\}_{i\in\{1,\ldots,n\}}}\ \textsc{T-Offer}$$

$$\frac{\Delta;\Gamma, x^p : S_i\{B/X_i\} \vdash P \quad \Delta \vdash T \leqslant \oplus\langle l_i(X_i \leqslant B_i) : S_i\rangle_{i\in\{1,\ldots,n\}} \quad 1 \le i \le n \quad \Delta \vdash B \leqslant B_i}{\Delta;\Gamma, x^p : T \vdash x^p \triangleleft l_i(B)\,.\,P}\ \textsc{T-Choose}$$

Figure 2: Typing Rules

time. In T-Offer, there are a number of potential continuation processes depending on the label received. Although the type variables $X_1,\ldots,X_n$ are bound in the offer construct, each P_i has X_i free and must be well typed in an environment where the type variable X_i and its upper bound B_i are added to the Δ component. In the final rule, T-Choose, the environment that types the continuation process P is more complicated. Each type S_i contains a type variable X_i. P does not though, as it uses the chosen type B that is passed with the label. P, therefore, must be typable in an environment where the session port has type S_i with B substituted for X_i. Finally, the type B must be a subtype of the appropriate upper bound, B_i.

For the purposes of the examples in this paper we assume a language for arithmetic expressions involving typing rules such as the following.

$$\frac{\Delta;\Gamma \vdash a : X \quad \Delta;\Gamma \vdash b : X \quad \Delta \vdash X \leqslant \mathsf{real}}{\Delta;\Gamma \vdash a + b : X}\ \textsc{T-Plus}$$

More generally, we could extend the language to include constructors such as product types, record types and function types with appropriate subtyping rules, giving a much richer language.

5 Operational Semantics

As usual for languages based on the π-calculus, the operational semantics is defined using a reduction relation [12]. $P \xrightarrow{x,l(B)} Q$ means that process P reduces to process Q in a single step, either by means of a communication, a choice or a conditional statement. Where the reduction involves a communication or choice, x is the channel on which this takes place. Only if the reduction is a choice, $l(B)$ is the label, l, which is chosen and the type, B, that is passed with the label. In all other reductions $x = \tau$ and $l(B) = \_$.

Reductions are labelled in this way for two reasons: Firstly, where the reduction is on a channel bound by a ν, we need to know this information to change the type in the ν. (This occurs in the R-NewX rule described below.) Additionally, it is needed for the statement and the proof of the subject reduction theorem, also described below. The labels have no semantic significance and would be omitted in any implementation.

$$\frac{p = \bar{q}}{x^p\ ?\ [\tilde{y} : \tilde{T}]\ .\ P \mid x^q\ !\ [\tilde{z}]\ .\ Q \xrightarrow{x,\_} P\{\tilde{z}/\tilde{y}\} \mid Q}\ \textsc{R-Comm}$$

$$\frac{i \in \{1, \ldots, n\} \quad p = \bar{q}}{x^p \triangleright \{l_i(X_i \leqslant B_i) : P_i\}_{i \in \{1,\ldots,n\}} \mid x^p \triangleleft l_i(B)\ .\ Q \xrightarrow{x,l_i(B)} P_i\{B/X_i\} \mid Q}\ \textsc{R-Select}$$

$$\frac{}{\textsf{if}\ \mathit{true}\ \textsf{then}\ P\ \textsf{else}\ Q \xrightarrow{\tau,\_} P}\ \textsc{R-True} \qquad \frac{}{\textsf{if}\ \mathit{false}\ \textsf{then}\ P\ \textsf{else}\ Q \xrightarrow{\tau,\_} Q}\ \textsc{R-False}$$

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

$$\frac{P \xrightarrow{\alpha,l(V)} P' \quad \alpha \neq x}{(\nu x^{\pm} : S)P \xrightarrow{\alpha,l(V)} (\nu x^{\pm} : S)P'}\ \textsc{R-New} \qquad \frac{P \xrightarrow{x,l(V)} P'}{(\nu x^{\pm} : S)P \xrightarrow{\tau,\_} (\nu x^{\pm} : \mathit{tail}(S, l(V)))P'}\ \textsc{R-NewX}$$

Figure 3: Reduction Rules

Our reduction relation is the smallest relation closed under the rules in Figure 3. R-Comm is the standard rule for communication, where the names received are substituted in the continuation process P. Note that we have a condition in the hypothesis that the session ports must have dual polarities. R-Select is the rule for selection from a choice of labels. A reduction is possible if the label chosen is one of those offered and, again, if the ports have dual polarities. Note here that a type substitution occurs in the continuation of the process offering the choice. R-True and R-False are standard, defining reduction in conditional expressions. R-Par and R-Cong are also standard, defining reduction under parallel composition and structural congruence. Finally, we have two reduction rules for processes under ν bindings. R-New, allows reductions under a ν binding where the channel, x, on which the reduction takes place is not the name being bound. R-NewX, allows reductions under a ν binding where the reduction is on the channel being bound. In this case, the resulting process has a ν binding with a new type for the channel given by applying the *tail* function to the old type, where *tail* is defined as follows.

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

We prove type soundness in the usual way: We prove a subject reduction theorem demonstrating that a well-typed process with a reduction will reduce to another well-typed process. Then we prove that any possible reductions immediately possible in a well-typed process do not cause errors. Together, these results imply that a well-typed process can reduce safely through any sequence of reduction steps.

We only prove type soundness for processes where the environment is balanced. This is because we are only interested in reductions resulting from the execution of programs. Although it is possible to type processes with unbalanced environments, for these to form part of a program the typing derivation would eventually have to balance the environment for the channels in it to be bound by a ν.

Theorem 1 (Subject Reduction) *If $\Gamma \vdash P$, $P \xrightarrow{\alpha,l(B)} P'$ and Γ is balanced then $\Gamma' \vdash P'$ and Γ' is balanced.*

Proof: By induction on the derivation of $\Gamma \vdash P$. The assumptions that $\Gamma \vdash P$ is derivable and Γ is balanced provide the information about the components of P' which is needed to build a derivation of $\Gamma' \vdash P'$.

Theorem 2 (Run-Time Safety) *If $\Delta; \Gamma \vdash P$, $P \equiv (\nu\tilde{v} : \tilde{V})(Q \mid R)$ and $\Delta; \Gamma$ is balanced, then*

1. *if $Q \equiv x^p\, ?\, [\tilde{y} : \tilde{T}]\, .\, Q_1 \mid x^q\, !\, [z]\, .\, Q_2$ then $\forall z_i \in \tilde{z}. \Delta; \Gamma, \tilde{v} : \tilde{V} \vdash z_i : U_i$ where $U_i \leqslant T_i$*
2. *if $Q \equiv x^p \triangleright \{l_i(X_i \leqslant B_i) : Q_i\}_{i\in\{1,\ldots,n\}} | x^q \triangleleft l_i(B)\, .\, S$ then $l_i \in \{l_1, \ldots, l_n\}$ and $\Delta \vdash B \leqslant B_i$*
3. *if $Q \equiv \mathsf{if}\ x\ \mathsf{then}\ P\ \mathsf{else}\ Q$ then $\Delta; \Gamma, \tilde{v} : \tilde{V} \vdash x : T$ where $T \leqslant \mathsf{bool}$*

Proof: From the derivation of $\Delta; \Gamma \vdash P$. On reconstructing the typing derivation for each of the potential reductions the typing rules yield the information necessary to construct the relevant conclusion.

6 Example

In this section we give an example of a typing derivation and a sequence of reductions for a process. The process is the addition interaction from Section 1.

6.1 Typing Derivation

Here we derive

$$\emptyset; x^+ : S, x^- : \overline{S}, 11 : \mathsf{int}, 7 : \mathsf{int} \vdash x^+ \triangleright \{\mathsf{plus}(X \leqslant \mathsf{real}) : P\} \mid x^- \triangleleft \mathsf{plus}(\mathsf{int})\, .\, Q$$

where

$$\begin{array}{rcl} S & = & \&\langle \mathsf{plus}(X \leqslant \mathsf{real}) : T \rangle \\ T & = & ?[X] \,.\, ?[X] \,.\, ![X] \,.\, \mathsf{end} \\ P & = & x^+ \,?\, [a : X] \,.\, x^+ \,?\, [b : X] \,.\, x^+ \,!\, [a+b] \,.\, \mathbf{0} \\ Q & = & x^- \,!\, [7] \,.\, x^- \,!\, [11] \,.\, x^- \,?\, [r : \mathsf{int}] \,.\, \mathbf{0} \end{array}$$

First we derive P, the body of the polymorphic plus operation. Here we have a single type variable X with an upper bound real, so $\Delta = X \leqslant \mathsf{real}$.

$$(1)\ \left\{ \dfrac{\dfrac{\Delta; x^+ : \mathsf{end}\ \text{completed}}{\Delta; x^+ : \mathsf{end} \vdash \mathbf{0}}\,\textsc{T-Nil} \qquad \Delta \vdash ![X] \,.\, \mathsf{end} \leqslant ![X] \,.\, \mathsf{end}}{\Delta; x^+ : ![X] \,.\, \mathsf{end}, a+b : X \vdash x^+ \,!\, [a+b] \,.\, \mathbf{0}}\,\textsc{T-Out} \right.$$

$$\dfrac{\dfrac{(1) \qquad \Delta \vdash ?[X] \,.\, ![X] \,.\, \mathsf{end} \leqslant ?[X] \,.\, ![X] \,.\, \mathsf{end}}{\Delta; x^+ : ?[X] \,.\, ![X] \,.\, \mathsf{end}, a : X \vdash x^+ \,?\, [b : X] \,.\, x^+ \,!\, [a+b] \,.\, \mathbf{0}}\,\textsc{T-In} \qquad \Delta \vdash T \leqslant T}{\Delta; x^+ : T \vdash P}\,\textsc{T-In}$$

Then we derive Q, the body of the client that has integers to be added.

$$(2)\ \left\{ \dfrac{\dfrac{\emptyset; x^- : \mathsf{end}, r : \mathsf{int}\ \text{completed}}{\emptyset; x^- : \mathsf{end}, r : \mathsf{int} \vdash \mathbf{0}}\,\textsc{T-Nil} \qquad \emptyset \vdash ?[\mathsf{int}] \,.\, \mathsf{end} \leqslant ?[\mathsf{int}] \,.\, \mathsf{end}}{\emptyset; x^- : ?[\mathsf{int}] \,.\, \mathsf{end} \vdash x^- \,?\, [r : \mathsf{int}] \,.\, \mathbf{0}}\,\textsc{T-In} \right.$$

$$(3) \qquad \emptyset \vdash \overline{T}\{\mathsf{int}/X\} \leqslant \overline{T}\{\mathsf{int}/X\}$$

$$\dfrac{\dfrac{(2) \qquad \emptyset \vdash ![\mathsf{int}] \,.\, ?[\mathsf{int}] \,.\, \mathsf{end} \leqslant ![\mathsf{int}] \,.\, ?[\mathsf{int}] \,.\, \mathsf{end}}{\emptyset; x^- : ![\mathsf{int}] \,.\, ?[\mathsf{int}] \,.\, \mathsf{end}, 11 : \mathsf{int} \vdash x^- \,!\, [11] \,.\, x^- \,?\, [r : \mathsf{int}] \,.\, \mathbf{0}}\,\textsc{T-Out} \qquad (3)}{\emptyset; x^- : \overline{T}\{\mathsf{int}/X\}, 11 : \mathsf{int}, 7 : \mathsf{int} \vdash Q}\,\textsc{T-Out}$$

Finally, we add the offer construct to P and the choose construct to Q. The type variable X is bound in the offer construct, the result being that both processes now have no free type variables and can be put in parallel.

$$(4) \qquad \left\{ \dfrac{\Delta; x^+ : T \vdash P \quad \emptyset \vdash S \leqslant S}{\emptyset; x^+ : S \vdash x^+ \triangleright \{\mathsf{plus}(X \leqslant \mathsf{real}) : P\}}\,\textsc{T-Offer} \right.$$

$$(5) \quad \left\{ \dfrac{\emptyset; x^- : \overline{T}\{\mathsf{int}/X\}, 11 : \mathsf{int}, 7 : \mathsf{int} \vdash Q \quad \emptyset \vdash \overline{S} \leqslant \overline{S} \quad \emptyset \vdash \mathsf{int} \leqslant \mathsf{real}}{\emptyset; x^- : \overline{S}, 11 : \mathsf{int}, 7 : \mathsf{int} \vdash x^- \triangleleft \mathsf{plus}(\mathsf{int}) \,.\, Q}\,\textsc{T-Choose} \right.$$

$$\dfrac{(4) \qquad (5)}{\emptyset; x^+ : S, x^- : \overline{S}, 11 : \mathsf{int}, 7 : \mathsf{int} \vdash x^+ \triangleright \{\mathsf{plus}(X \leqslant \mathsf{real}) : P\} \mid x^- \triangleleft \mathsf{plus}(\mathsf{int}) \,.\, Q}\,\textsc{T-Par}$$

6.2 Reduction Steps

There are four reduction steps in the execution of the process. The first is an application of the R-Select rule, the other three being applications of the R-Comm rule.

$(a)\quad x^+ \triangleright \{\mathsf{plus}(X \leqslant \mathsf{real}) : P\} \mid x^- \triangleleft \mathsf{plus}(\mathsf{int}) \,.\, Q \xrightarrow{x,\mathsf{plus}(\mathsf{int})} P\{\mathsf{int}/X\} \mid Q$

$(b)\quad x^+\ ?\ [a : \mathsf{int}] \,.\, x^+\ ?\ [b : \mathsf{int}] \,.\, x^+\ !\ [a + b] \,.\, \mathbf{0} \mid x^-\ !\ [7] \,.\, x^-\ !\ [11] \,.\, x^-\ ?\ [r : \mathsf{int}] \,.\, \mathbf{0}$
$\xrightarrow{x,-} x^+\ ?\ [b : \mathsf{int}] \,.\, x^+\ !\ [a + b] \,.\, \mathbf{0} \mid x^-\ !\ [11] \,.\, x^-\ ?\ [r : \mathsf{int}] \,.\, \mathbf{0}$

$(c)\quad x^+\ ?\ [b : \mathsf{int}] \,.\, x^+\ !\ [a + b] \,.\, \mathbf{0} \mid x^-\ !\ [11] \,.\, x^-\ ?\ [r : \mathsf{int}] \,.\, \mathbf{0}$
$\xrightarrow{x,-} x^+\ !\ [a + b] \,.\, \mathbf{0} \mid x^-\ ?\ [r : \mathsf{int}] \,.\, \mathbf{0}$

$(d)\quad x^+\ !\ [a + b] \,.\, \mathbf{0} \mid x^-\ ?\ [r : \mathsf{int}] \,.\, \mathbf{0} \xrightarrow{x,-} \mathbf{0}$

7 Conclusions and Future Work

We have identified a problem with the use of subtyping in combination with session types in the π-calculus in that it is only possible for a client to take advantage of this subtyping in relation to messages sent to the server, not those received from it. We have proposed a system of bounded polymorphism incorporating session types which solves this problem by introducing a dependency between the type of a message from the client (a selection) and the types of messages which come from the server after that selection has been made.

7.1 Related Work

Polymorphism (in the style of the polymorphic lambda calculus (System F) [7, 19]) has been studied in the π- calculus by Turner [22] and Pierce and Sangiorgi [17]. The programming language Pict, based on the π- calculus, has a polymorphic type system. Weaker ML-style polymorphism has been studied in π-calculus-like languages by Vasconcelos and Honda [24] and the first author [2]. A rather different style of polymorphism has been proposed by Liu and Walker [11]. We believe that the present paper is the first study of bounded polymorphism in the π- calculus, and the first study of polymorphism in relation to session types.

7.2 Further Work

In the syntax of processes we have restricted our upper bounds to be datatypes or other type variables. This may seem like an arbitrary restriction, but the reason for this is as follows: It is the case that if $T \leqslant U$ then $\overline{U} \leqslant \overline{T}$. One result of this is that if we have a type variable X with a session type as its upper bound, say $?[\mathsf{bool}] \,.\, \mathsf{end}$, we have a lower bound for $\overline{X}$, i.e. $![\mathsf{bool}] \,.\, \mathsf{end}$, but no upper bound. With no upper bound, it is not possible to construct a typing derivation for processes using $\overline{X}$, and so no programs using session types as upper bounds would be typeble, even if the syntax of processes allowed it.

One possible solution may be to include a lower and upper bound on type variables i.e. $B_1 \leqslant X \leqslant B_2$. This would provide the necessary lower and upper bounds for dual types: $\overline{B_2} \leqslant \overline{X} \leqslant \overline{B_1}$. It would be necessary to find some motivating examples for this, however, and it may be the case that there are no useful classes of process that we would want to be typable with session types as upper bounds. Clearly there is scope for further work in this area.

Our definition of subtyping for branch and choice types resembles kernel $F_{<:}$in that the upper bounds do not change when you go up the subtyping relation. We could investigate a system more similar to full $F_{<:}$, but again would need some motivating examples. Finally, we could add normal channels and investigate bounded polymorphism as an extension to the type system of the Pict [18] programming language, with type variable bindings separated from the offer construct.

Acknowledgements

This research was funded by the EPSRC project "Novel Type Systems for Concurrent Programming Languages" (GR/L75177,GR/N39494). Paul Taylor's proof tree macros were used in the production of this report.

References

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

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

[3] S. J. Gay and M. J. Hole. Types and subtypes for client-server interactions. In S. D. Swierstra, editor, *ESOP'99: Proceedings of the European Symposium on Programming Languages and Systems*, volume 1576 of *Lecture Notes in Computer Science*, pages 74–90. Springer-Verlag, 1999.

[4] S. J. Gay and M. J. Hole. Types for correct communication in client-server systems. Technical Report CSD-TR-00-07, Department of Computer Science, Royal Holloway, University of London, 2000.

[5] S. J. Gay and M. J. Hole. Types and subtypes for correct communication in client-server systems. Technical Report TR-2003-131, Department of Computing Science, University of Glasgow, February 2003.

[6] S. J. Gay, A. Ravara, and V. T. Vasconcelos. Session types for inter-process communication. Technical Report TR-2003-133, Department of Computing Science, University of Glasgow, March 2003.

[7] J.-Y. Girard. *Interprétation fonctionelle et élimination des coupures dans l'arithmétique d'ordre supérieur*. PhD thesis, University of Paris VII, 1972.

[8] K. Honda, V. Vasconcelos, and M. Kubo. Language primitives and type discipline for structured communication-based programming. In C. Hankin, editor, *ESOP'98: Proceedings of the European Symposium on Programming*, volume 1381 of *Lecture Notes in Computer Science*, pages 122–138. Springer-Verlag, 1998.

[9] N. Kobayashi, B. C. Pierce, and D. N. Turner. Linearity and the pi-calculus. In *Proceedings, 23rd ACM Symposium on Principles of Programming Languages*, 1996.

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

[11] X. Liu and D. Walker. A polymorphic type system for the polyadic π-calculus. In *CONCUR'95: Proceedings of the International Conference on Concurrency Theory*, volume 962 of *LNCS*. Springer-Verlag, 1995.

[12] R. Milner. The polyadic π-calculus: A tutorial. Technical Report 91-180, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, 1991.

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

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

[15] B. C. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5), 1996.

[16] B. C. Pierce. *Types and Programming Languages*. MIT Press, 2002.

[17] B. C. Pierce and D. Sangiorgi. Behavioral equivalence in the polymorphic pi-calculus. In *Proceedings, 24th ACM Symposium on Principles of Programming Languages*, 1997.

[18] B. C. Pierce and D. N. Turner. Pict: A programming language based on the pi-calculus. Technical Report CSCI 476, Computer Science Department, Indiana University, 1997. In *Proof, Language and Interaction: Essays in Honour of Robin Milner*, Gordon Plotkin, Colin Stirling, and Mads Tofte, editors, MIT Press, 2000.

[19] J. C. Reynolds. Towards a theory of type structure. In *Paris colloquium on programming*, volume 19 of *LNCS*. Springer-Verlag, 1974.

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

[21] K. Takeuchi, K. Honda, and M. Kubo. An interaction-based language and its typing system. In C. Halatsis, D. G. Maritsas, G. Philokyprou, and S. Theodoridis, editors, *PARLE '94: Parallel Architectures and Languages Europe, 6th International PARLE Conference, Proceedings*, volume 817 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.

[22] D. N. Turner. *The Polymorphic Pi-Calculus: Theory and Implementation*. PhD thesis, University of Edinburgh, 1996.

[23] A. Vallecillo, V. T. Vasconcelos, and A. Ravara. Typing the behavior of objects and components using session types. In *1st International Workshop on Foundations of Coordination Languages and Software Architectures (FOCLASA 2002)*, volume 68 of *Electronic Notes in Theoretical Computer Science*. Elsevier, August 2002.

[24] V. T. Vasconcelos and K. Honda. Principal typing schemes in a polyadic π-calculus. In *CONCUR'93: Proceedings of the International Conference on Concurrency Theory*, Lecture Notes in Computer Science. Springer-Verlag, 1993.