Structured Communications with Concurrent Constraints

Mario Coppo and Mariangiola Dezani-Ciancaglini

Dipartimento di Informatica, Università di Torino

Abstract. We propose a calculus which combines *concurrent constraints*, *name passing* and *sessions*. In this way we get enough expressivity to represent both quality of services and safety of interaction between clients and servers. Central for the soundness of our calculus is a type assignment system whose main novelty is the assurance of *channel bilinearity* in presence of channel constraints, channel delegations and processes recursions.

1 Introduction

Service Level Agreements are expected to specify, in Service Oriented Computing, both the client requirements, the service guarantees and the quality of services (cost, availability, etc..). Moreover after the client and the server have started to communicate it is essential to assure that both behave as expected, in order to guarantee the safety of the interaction. A case in point is delegation of activities to third parties, which often occurs transparently to the client.

This paper proposes a calculus to describe contracts which combines different programming paradigms to achieve these goals. We use the notion of *concurrent constraint programming* [8], enriched with a *name-passing* (aliasing) mechanism [7], in order to explicitly represent, through the notion of constraint, relations involving private and public names (see [3, 4]). In this calculus data communication is symmetric in input and output and is achieved via the introduction of constraints between channel names. Public and private constraints specify the requirements, from both the customer and the provider side, to open new interactions and to conduct them.

Moreover we exploit the notion of session and session type [9, 6] to design communication protocols which assure safe and reliable communication sequences. A *session* is an abstraction of a series of communications between two processes. In our calculus the opening of sessions is controlled by a set of constraints that specify the requirements imposed by both the server and the client to start the communication. Standard `tell`, `check` and `ask` primitives allow further control and conditional branching while sessions are carried on. A branching primitive based on label exchange is also included in the language.

The resulting calculus combines the flexibility and the expressive power of constraint and name-passing programming with the safeness of communications guaranteed by session types. A relevant feature of our approach is the primitive for opening sessions, which allows to specify a set of constraints whose satisfaction is necessary for starting the session interaction. Usually in sessions the privacy of communications is assured by creating a new fresh channel restricted to the participants. In our calculus instead we obtain this through a constraint that explicitly associates the client and

the server channels making them private and exclusively reserved to carry on the communication prescribed by the session. As usual in the literature on session types this condition is called *bilinearity*. To increase the expressive power of the calculus we also introduce a delegation primitive which allows to involve other agents in a communication protocol transferring in a safe way the communication sequence to appropriate partners.

A type assignment system assures both the well formedness of processes and constraints and the correctness and bilinearity of the communication protocols. Being all communications defined in terms of constraints we need a particular care in the definition of typing rules to assure the bilinearity condition in presence of delegation and recursive processes. In fact we cannot simply state that each channel name must occur exactly once, since channel delegation and process recursion need propagation of constraints. Our solution is to control via the type system the number of possible occurrences of channel names in constraints. In particular we allow channel names to occur at most twice in constraints, but we assure at the same time that all channels which can communicate appear at most once in constraints. This is possible since constraints between channel names can only be generated by the reduction rules. The programmer is only allowed to tell, check and ask constraints between ground expressions.

A subject reduction theorem proves that the property of being well typed is preserved along reduction, showing the consistency and reliability of the system.

Related work Our calculus is an enrichment of π-calculus [7] with primitives for handling communication protocols whose execution can start only if the conditions required by the participants can be satisfied. From π-calculus we inherit the possibility of communicating and restricting names.

For describing communication protocols we essentially follow [9, 6], the main difference being that a session opening is conditioned by a set of constraints and that the communication of data is realised by means of name fusion [10].

Our primitives for testing and adding constraints are a subset of those proposed in [3, 4]. We consider only a simple notion of constraint including equalities of names, equalities and order relations between ground expressions. More general notions of constraints, like in [3] or [1], could also be considered, but we leave this for future work. One novelty of our approach is that we have also constraints between channels which need to be used bilinearly in order to guarantee the safety of sessions. Channel bilinearity in presence of delegation needs a careful treatment in order to preserve types under reduction as discussed in [11]. Instead for data different from channels our constraints are a particular case of the general treatment of [3, 4]. Our choice is motivated by the desire of modelling the interaction between constraints and sessions without introducing orthogonal features.

A related paper is also [2], where traditional sessions are made more robust by adding correspondence assertions.

Paper Structure Section 2 describes the syntax and the operational semantics of our calculus, illustrating them also by means of an example. Section 3 discusses the type assignment system and Section 4 the features of well-typed processes. Section 5 draws some future work directions. For referee's convenience only the appendix contains the proofs.

2 The Calculus

2.1 Syntax

	constraints		
c	::=	$e = e' \mid e \leq e'$	*simple*
C	::=	$\mathsf{c} \mid C \mid C$	*composed*
	pure processes		
P,Q	::=	$x\{C\}(y).P$	*session opening*
	$\mid$	$y\langle(x)\rangle.P$	*data communication*
	$\mid$	$y\langle z\rangle.P$	*channel delegation*
	$\mid$	$y(z).P$	*channel reception*
	$\mid$	$y \oplus l.P$	*label selection*
	$\mid$	$y\&\{\overrightarrow{l : P}\}$	*label branching*
	$\mid$	$\mathbf{0}$	*inaction*
	$\mid$	$P \mid Q$	*parallel composition*
	$\mid$	$(x)P$	*data name restriction*
	$\mid$	$\mathsf{def}\ D\ \mathsf{in}\ P$	*recursive definition*
	$\mid$	$X\langle e,z\rangle$	*process call*
	$\mid$	$\mathtt{tell}(\mathsf{c}).P$	*tell*
	$\mid$	$\mathtt{check}(\mathsf{c}).P$	*check*
	$\mid$	$\mathtt{ask}(\overrightarrow{\mathsf{c} : P})$	*alternative ask*
	constrained processes		
S,R	::=	P	*pure process*
	$\mid$	c	*constraint*
	$\mid$	$S \mid R$	*parallel composition*
	$\mid$	$(v)\,S$	*restriction*
	$\mid$	$\mathsf{def}\ D\ \mathsf{in}\ S$	*recursive definition*
	auxiliary definitions		
v	::=	$x \mid y \mid z \mid \ldots$	*names*
e	::=	$v \mid \mathsf{true} \mid \mathsf{false} \mid 0 \mid 1 \mid \ldots \mid e + e \mid e * e \mid \ldots$	*expressions*
D	::=	$X(x,y) = P$	*declaration for recursion*

Table 1. Syntax

Our calculus is an extension of the π-calculus [7] inspired by the session calculus of [6] and by the concurrent constraint calculus of [3].

Table 1 gives the syntax of the calculus. The *simple constraints* we consider are only equalities between names and equalities and order relations between expressions of ground types (boolean and natural). Meaningful constraints can only equate expressions of the same type and compare naturals with respect to their order. This will be assured by the type system of next section. Clearly there is no problem in changing or extending the set of expressions and the kinds of constraints, provided that a suitable notion of provability and consistency (see Subsection 2.2) can be defined on them. *Composed constraints* are parallels of simple constraints.

In this table x, y, z,.. denote names which can be free (public) or restricted (private). We convene that x ranges over data names, i.e. session names or names of ground expressions, while y, z range over communication channels names. We use v to denote

both kinds of names. An overlined name $\overline{v}$ is meaningful only if v does not stand for a ground expression. In this case v represents the dual session name or the dual channel name of v. Unless explicitly stated, a name v is used to denote both v and $\overline{v}$. The duality is an involution, i.e. $\overline{\overline{v}} = v$.

The *pure processes* contain constraints only inside prefixes. In the prefix $x\{C\}(y)$ for session opening, x is the session name and y is the associated private channel. The set C is the set of constraints controlling the opening of sessions. The following five prefixed processes denote different kinds of communication on the channel y, called the *subject* of the communication. Communication of data names (realised by name fusion) does not require to distinguish between input and output, so we have an unique prefix. Instead, as we will see in Section 3, to preserve the bilinearity of communications we need to distinguish channel delegation (in which z is not restricted) from channel reception (in which z is restricted). In label selection and branching l ranges over a distinguished set of labels. The selection chooses an alternative and the branching offers different alternatives, as in [6].

The last three prefixed processes are take from [3, 1] and dealt with constraints. Both `tell` and `check` test the consistency of a constraint with the current set of constraints, and if the test is successful `tell` adds the constraint. Instead `ask` allows different alternatives controlled by the corresponding constraints. The remaining constructors for pure processes are standard in π-calculus [7].

The syntax of *constrained processes* put in parallel pure processes and constraints.

Free and restricted names for constraints and processes are defined as usual, convening that:

- a name v occurs in a constrained process S when v or $\overline{v}$ occurs in S;
- a restriction for v bounds both v and $\overline{v}$;
- x and y are restricted in the declaration $X(x,y) = P$.

We denote by $fn(S)$ the set of names which occur free in S, and similarly for D. Process variables are bound by recursive definitions: we define $dpv(D)$=X if D is $X(x,y) = P$ and we denote by $fpv(S)$ the set of process variables which occur free in S.

A channel y is *active* in a constrained process S if y occurs in S inside a pure process.

We use the $\overrightarrow{\xi}$ to denote a sequence $\xi_1,\ldots,\xi_n$. We also write $\xi \in \overrightarrow{\xi}$ to denote that $\xi = \xi_i$ for some $(1 \leq i \leq n)$. We will use $[]$ to separates alternatives in branching and ask. So, for instance, $\{\overrightarrow{l;T}\}$ stands for $\{l_1 : T_1 \,[]\, \ldots \,[]\, l_n : T_n\}$.

2.2 Operational Semantics

Since processes contain constraints which determine reduction, the semantics is given in three steps: structural equivalence, provability/consistency of constraints and reduction rules.

Structural equivalence Constrained processes are considered modulo α-conversion of restricted names and the structural rules of Table 2, which are standard.

Owing to structural equivalence each constrained process can be written in *canonical form*, i.e. as the parallel composition of a set of constraints with a pure process restricted with respect to the common names:

$$\overrightarrow{(v)}(C \mid P).$$

$$S \mid \mathbf{0} \equiv S \quad S \mid R \equiv R \mid S \quad (S \mid R) \mid R' \equiv S \mid (R \mid R') \quad (v)S \mid R \equiv (v)(S \mid R) \quad \text{if } v \notin fn(R)$$

$$(v)(v')S \equiv (v')(v)S \quad (v)\mathbf{0} \equiv \mathbf{0} \quad \mathsf{def}\ D\ \mathsf{in}\ \mathbf{0} \equiv \mathbf{0} \quad \mathsf{def}\ D\ \mathsf{in}\ (v)S \equiv (v)\mathsf{def}\ D\ \mathsf{in}\ S \quad \text{if } v \notin fn(D)$$

$$(\mathsf{def}\ D\ \mathsf{in}\ S) \mid R \equiv \mathsf{def}\ D\ \mathsf{in}\ (S \mid R) \quad \text{if } dpv(D) \notin fpv(R)$$

$$\mathsf{def}\ D\ \mathsf{in}\ (\mathsf{def}\ D'\ \mathsf{in}\ S) \equiv \mathsf{def}\ D'\ \mathsf{in}\ (\mathsf{def}\ D\ \mathsf{in}\ S) \quad \text{if } dpv(D) \neq dpv(D')$$

Table 2. Structural equivalence

$$\frac{}{C \mid \mathsf{c} \vdash_c \mathsf{c}}\ [\text{Axiom}] \qquad \frac{}{C \vdash_c e = e}\ [\text{Ident}] \qquad \frac{}{C \vdash_c \overline{\overline{x}} = x}\ [\text{Invol}]$$

$$\frac{C \vdash_c x = y}{C \vdash_c \overline{x} = \overline{y}}\ [\text{Dual}] \qquad \frac{C \vdash_c e_1 = e_2}{C \vdash_c e_2 = e_1}\ [\text{Simm}] \qquad \frac{C \vdash_c e1 \ = \ e_2 \qquad C \vdash_c e_2 \ = \ e_3}{C \vdash_c e_1 \ = \ e_3}\ [\text{Trans}]$$

Table 3. Constraint inference

Provability and consistency of constraints Table 3 gives the interesting rules for proving that a constraint c is a logical consequence of a set of constraints C (notation $C \vdash_c \mathsf{c}$). To prove constraints about boolean and arithmetic expressions we assume to have also a set of rules for standard elementary boolean and arithmetic reasoning, which are omitted here.

The rules of Table 3 are complete for proving equality and duality of names. The rules [Invol] and [Dual] give the desired meaning to overlined names. In all cases both $C \vdash_c \mathsf{c}$ and its negation $C \not\vdash_c \mathsf{c}$ are easily shown to be decidable.

A set of constraint is *consistent* (notation *cons*(C)) if no contradiction can be derived from it by the rules of Table 3 plus those for boolean and arithmetic expressions. So also consistency is decidable.

Reduction rules The reduction rules for the operational semantics are given in Table 4. We can assume, by rule [Str], that each process is in canonical form. This allows us to require that the collection C of constraints occurring in the reduction rules represent all the constraints in the system. This condition is enforced by rule [Par], which allows to put redexes and contracta in parallel with pure processes, but not with constraints. Via the structural rules [Defin], [Scop] and [Par] we can extend the reduction relation to all reducible contexts.

Rule [Link] describes the initialization of a new session between two participants with dual session names, provided both the constraints C_1 and C_2 required to activate the respective participants are consistent with the global constraints C. Note that the duality of session names could have been obtained via some previous communication. When the session starts, the constraints C_1 and C_2 are added to the set of global constraints: in this way we assure they will be true for the execution of the whole session. The participants will communicate through the private channels y_1, y_2. The constraint $y_1 = \overline{y_2}$ declares that the two channels are dual.

Rule [Comm] allows communication via fusion on two dual channels for data exchange. Note that there is no distinction between input and output, but a constraint forcing equality between x_1 and x_2, if it is consistent with C, is generated.

$$C \mid x_1\{C_1\}(y_1)P_1 \mid x_2\{C_2\}(y_2)P_2 \longrightarrow C \mid C_1 \mid C_2 \mid (y_1)(y_2)(y_1 = \overline{y_2} \mid P_1 \mid P_2)$$
$$\text{if } C \vdash_c x_1 = \overline{x_2} \text{ and } cons(C \mid C_1 \mid C_2) \text{ and } y_1 \notin fn(P_2) \text{ and } y_2 \notin fn(P_1) \qquad \text{[Link]}$$

$$C \mid y_1\langle\!\langle x_1 \rangle\!\rangle.P_1 \mid y_2\langle\!\langle x_2 \rangle\!\rangle.P_2 \longrightarrow C \mid x_1 = x_2 \mid P_1 \mid P_2 \quad \text{if } C \vdash_c y_1 = \overline{y_2} \text{ and } cons(C \mid x_1 = x_2) \qquad \text{[Comm]}$$

$$C \mid y_1\langle z_1 \rangle.P_1 \mid y_2(z_2).P_2 \longrightarrow C \mid P_1 \mid (z_2)(z_1 = z_2 \mid P_2) \text{ if } C \vdash_c y_1 = \overline{y_2} \qquad \text{[Deleg]}$$

$$C \mid y_1 \& \overrightarrow{\{l : P\}} \mid y_2 \oplus l_i.P' \longrightarrow C \mid P_i \mid P' \text{ if } C \vdash_c y_1 = \overline{y_2} \text{ and } l_i \in \overrightarrow{l} \qquad \text{[Branch]}$$

$$C \mid \mathtt{tell}(\mathsf{c}).P \longrightarrow C \mid \mathsf{c} \mid P \quad \text{if } cons(C \mid \mathsf{c}) \qquad \text{[Tell]}$$

$$C \mid \mathtt{check}(\mathsf{c}).P \longrightarrow C \mid P \text{ if } cons(C \mid \mathsf{c}) \qquad \text{[Check]}$$

$$C \mid \mathtt{ask}(\overrightarrow{\mathsf{c} : P}) \longrightarrow C \mid P_i \text{ if } C \vdash_c \mathsf{c}_i \text{ and for all } 1 \leq j < i \;\; C \not\vdash_c \mathsf{c}_j \qquad \text{[Ask]}$$

$$C \mid \mathsf{def}\, X(x,y) = P \;\mathsf{in}\; X(e,z) \longrightarrow C \mid \mathsf{def}\, X(x,y) = P \;\mathsf{in}\; (x)(y)(x{=}e \mid y{=}z \mid P) \qquad \text{[Rec]}$$

$$S \longrightarrow S' \Rightarrow \mathsf{def}\, D \;\mathsf{in}\; S \longrightarrow \mathsf{def}\, D \;\mathsf{in}\; S' \qquad \text{[Defin]}$$

$$S \longrightarrow S' \Rightarrow (v)S \longrightarrow (v)S' \quad S \longrightarrow S' \Rightarrow S \mid P \longrightarrow S' \mid P \qquad \text{[Scop,Par]}$$

$$S_1 \equiv S_2 \text{ and } S_2 \longrightarrow S_3 \text{ and } S_3 \equiv S_4 \Rightarrow S_1 \longrightarrow S_4 \qquad \text{[Str]}$$

Table 4. Reduction rules

In a channel delegation (rule [Deleg]) there is instead an explicit distinction between input and output represented by two different constructs. This is necessary since we want to preserve bilinearity, i.e. that each active channel has only one dual active channel. This is also the reason why we need to restrict the name accepting the delegated channel, in order to prevent the introduction of other constraints on it, thus leaving only the original channel visible from the outside. The type system will play an essential role in assuring safety of delegation.

Rule [Branch] selects the continuation process in the branch according to the label l_i as usual.

Constraints on channels are created by session opening, delegation, recursion, and tested in all communication rules. To assure channel bilinearity the type assignment system does not allow processes to add or check constraints involving channels. So in the following rules only constraints between data are considered. Rule [Tell] adds a new constraint provided that it is consistent with the set of global constraints and rule [Check] tests the consistency of a constraint against the set of global constraints. In rule [Ask] the continuation process P_i is the one associated with the first constraint c_i derivable from the set of global constraints.

The rule for evaluation of (recursive) definition is standard. Note that also the association of actual to formal parameters is realised via constraints.

2.3 Example

We illustrate our calculus through an example, which shows the interaction between constraints and sessions. The overall scenario involves a book Seller (S), two customers Alice (A) and Bob (B), and as global constraint the dollar/euro rate.

$$
\begin{array}{l}
\mathrm{S} = (stitle)(sprice)(saddress)(sordnum) \\
\quad (\overline{a}\{time = 3\}(y_1).y_1\langle\!\langle(stitle)\rangle\!\rangle.\texttt{tell}(sprice = \texttt{book-price-\$}).y_1\langle\!\langle(sprice)\rangle\!\rangle. \\
\quad y_1\&\{\mathsf{ok}.y_1\langle\!\langle(saddress)\rangle\!\rangle.\texttt{tell}(sordnum = \texttt{order-number}).y_1\langle\!\langle(sordnum)\rangle\!\rangle.\mathbf{0}[] \\
\qquad \mathsf{quit}.\mathbf{0}\}) \mid \\
\quad a\{time = 15\}(y_1)...\text{“}similar\ but\ with\ cheaper\ book\ price\text{”}) \\
\\
\mathrm{A} = (atitle)(aprice)(aaddress)(aordnum) \\
\quad (a\{time \leq 5\}(y_2).\texttt{tell}(atitle = \texttt{Hamlet}).y_2\langle\!\langle(atitle)\rangle\!\rangle.y_2\langle\!\langle(aprice)\rangle\!\rangle. \\
\quad \texttt{ask}((aprice * rate < 100€) : y_2 \oplus \mathsf{ok}.\texttt{tell}(aaddress = \texttt{Alice-address}). \\
\qquad\qquad y_2\langle\!\langle(aaddress)\rangle\!\rangle.y_2\langle\!\langle(aordnum)\rangle\!\rangle.\mathbf{0}[] \\
\qquad (\mathsf{true}) : (acontrib) \\
\qquad\quad (b\{title = atitle\}(z_1).z_1\langle\!\langle(acontrib)\rangle\!\rangle. \\
\qquad\quad \texttt{ask}\{(acontrib \geq aprice * rate\ div\ 2) : z_1 \oplus \mathsf{ok}.z_1\langle y_2\rangle.\mathbf{0}[] \\
\qquad\qquad (\mathsf{true}) : z_1 \oplus \mathsf{quit}.y_2 \oplus \mathsf{quit}.\mathbf{0}\})) \\
\\
\mathrm{B} = (bcontrib)(baddress)(bordnum) \\
\quad (\overline{b}\{title = \texttt{Hamlet}\}(z_2). \\
\quad \texttt{tell}(bcontrib = \texttt{offered-contrib}).z_2\langle\!\langle(bcontrib)\rangle\!\rangle. \\
\quad z_2\&\{\mathsf{ok}.z_2(t).t \oplus \mathsf{ok}.\texttt{tell}(baddress = \texttt{Bob-address}).t\langle\!\langle(baddress)\rangle\!\rangle.t\langle\!\langle(bordnum)\rangle\!\rangle.\mathbf{0}[] \\
\qquad \mathsf{quit}.\mathbf{0}\})
\end{array}
$$

Table 5. Bookseller example

The Seller provides a price list in dollars, and offers two delivery options via the session name $\overline{a}$: one assures reception of the book in 3 days and the other in 15 days. The former one is obviously more expensive. We have then two sessions offered by the Seller, with the same name, but with different initial constraints.

The buyer Alice chooses the first option but she is not sure to have enough euros to buy the book alone. So if the price is too high she asks Bob to participate in the business. The negotiation proceeds as follows:

- Alice and the Seller open a session using a and $\overline{a}$ and then Alice communicates the book title to the Seller. The Seller privately communicates to Alice the price in dollars of the book, which is converted in euro by Alice using the dollar/euro rate.
- If the price is within Alice's budget, she notifies the Seller (sending an ok label) that she accepts to buy the book. Then she communicates her address and receives the order number (via private constraints).
- If the price exceeds Alice's budget, she asks (through session b) for someone willing to contribute to buy the book. Her intention is expressed via a global constraint.
- Bob accepts to open the session on $\overline{b}$ (since he want to share the cost to buy the same book) and communicates to Alice how much he can contribute.
- If Bob'contribution covers at least half of the cost of the book, then Alice sends to Bob the ok label and delegates to him to conclude the session with the Seller as if he were Alice. Otherwise she communicates to both Bob and the Seller that she gives up (sending a quit label).

Notice that the delegation of Alice to Bob is transparent to the Seller.

The system is represented by the constrained process

$$(rate = 0.7) \mid \mathrm{S} \mid \mathrm{A} \mid \mathrm{B}$$

where S, A and B are defined in Table 5. It would be easy using process definition to modify the process S in such a way that the Seller offers his service in a permanent way.

3 Type System

3.1 Types

Sort	B	$::= \langle T \rangle \mid \mathsf{Bool} \mid \mathsf{Nat} \mid \ldots$	
Channel	T	$::= B;T$	*fusion*
		$\mid\ !ch(T);T$	*channel delegation*
		$\mid\ ?ch(T);T$	*channel reception*
		$\mid\ \oplus\{\overrightarrow{l:T}\}$	*selection*
		$\mid\ \&\{\overrightarrow{l:T}\}$	*branching*
		$\mid\ \mu\mathbf{t}.T$	*recursive*
		$\mid\ \mathbf{t}$	*variable*
		$\mid\ \mathsf{end}$	*end*
General	U	$::= B \mid T$	

Table 6. Syntax of Types

General types (given in Table 6) classify expressions in two disjoint kinds: data and channels. Data have *sort types*, i.e., *session types* $\langle T \rangle$ and *ground types* Bool, Nat, ... Session types associate channel types to session names.

Channel types describe the sequences of communications that take place during a session. The *fusion type* $B;T$ expresses that a names of type B is ready to be fused and then the session goes on according to T. The *channel delegation type* $!ch(T');T$ and the *channel reception type* $?ch(T');T$ express the sending and the receiving of a channel of type T', respectively. The *selection type* $\oplus\{\overrightarrow{l:T}\}$ represents the transmission of a label l_i chosen in the set $\overrightarrow{l}$ followed by the communications described in T_i. The *branching type* $\&\{\overrightarrow{l:T}\}$ represents the reception of a label l_i chosen in the set $\overrightarrow{l}$ followed by the communications described in T_i. Recursive types have the standard meaning and they are considered modulo fold/unfold, i.e. $\mu\mathbf{t}.T = T[\mu\mathbf{t}.T/\mathbf{t}]$. The end type signals the end of communications.

Each channel type T has a *dual channel type*, denoted by $\overline{T}$, which offers the complementary communications. The duality relation on channel types is defined by the following clauses, where $\dagger \in \{!,?\}$ and $\overline{!}=?$. Note that $\overline{\overline{T}}=T$.

$$\overline{B;T} = B;\overline{T} \qquad \overline{\dagger ch(T');T} = \overline{\dagger} ch(T');\overline{T} \qquad \overline{\mu\mathbf{t}.T} = \mu\mathbf{t}.\overline{T} \qquad \overline{\mathbf{t}} = \mathbf{t} \qquad \overline{\mathsf{end}} = \mathsf{end}$$

$$\overline{\oplus\{\overrightarrow{l:T}\}} = \&(\overrightarrow{\{l:\overline{T}\}}) \qquad \overline{\&\{\overrightarrow{l:T}\}} = \oplus(\overrightarrow{\{l:\overline{T}\}}).$$

3.2 Typing rules for expressions

Expressions can be both data names and channel names. For this reason we need type environments assigning types to both kinds of names. A crucial observation is that

$$\frac{}{\Gamma, x:B;\ \emptyset \vdash x:B}\ \lfloor \textsc{Data} \rfloor \qquad \frac{}{\Gamma, x:\langle T\rangle;\ \emptyset \vdash \overline{x}:\langle \overline{T}\rangle}\ \lfloor \textsc{SessDual} \rfloor$$

$$\frac{}{\Gamma;\ \{y:T\} \vdash y:T}\ \lfloor \textsc{Chann} \rfloor \qquad \frac{}{\Gamma;\ \{y:T\} \vdash \overline{y}:\overline{T}}\ \lfloor \textsc{ChannDual} \rfloor$$

$$\frac{}{\Gamma;\ \emptyset \vdash \mathsf{true}:\mathsf{Bool}}\ \lfloor \textsc{True} \rfloor \quad \frac{}{\Gamma;\ \emptyset \vdash \mathsf{false}:\mathsf{Bool}}\ \lfloor \textsc{False} \rfloor \quad \frac{}{\Gamma;\ \emptyset \vdash 0:\mathsf{Nat}}\ \lfloor \textsc{Zero} \rfloor \quad \ldots$$

$$\frac{\Gamma;\ \emptyset \vdash e:\mathsf{Nat} \quad \Gamma;\ \emptyset \vdash e':\mathsf{Nat}}{\Gamma;\ \emptyset \vdash e+e':\mathsf{Nat}}\ \lfloor \textsc{Plus} \rfloor \quad \ldots$$

Table 7. Typing rules for expressions

in order to preserve the bilinearity of channel names in pure processes each channel name can appear at most twice in well-typed constraints. We achieve this by taking two disjoint environments in the typing judgements for expressions. More precisely we define:

- *standard environments* (ranged over by Γ) which associate data names to sort types and process variables to pairs of sort types and channel types (the types of their parameters);
- *constraint session environments* (ranged over by Σ) which associate channel names to channel types and to $\bot$.

Formally:

$$\Gamma ::= \emptyset \mid \Gamma, x:B \mid \Gamma, X:BT \qquad \Sigma ::= \emptyset \mid \Sigma, y:T \mid \Sigma, y:\bot$$

where we assume that we can write $\Gamma, x:B$ only if x does not occur in Γ, briefly $x \notin dom(\Gamma)$ (we denote by $dom(\Gamma)$ the domain of Γ, i.e. the set of identifiers which occur in Γ) and similarly for Σ.

Note that *environments only contain assumptions for names without overline:* this is a key choice in order to guarantee that overlined names have dual types (see rules $\lfloor \textsc{SessDual} \rfloor$ and $\lfloor \textsc{ChannDual} \rfloor$ in Table 7). Session environments are relevant, i.e. they contain only the assumptions really used to prove the associated judgements (see rules $\lfloor \textsc{Chann} \rfloor$, $\lfloor \textsc{Channdual} \rfloor$).

The judgements for expressions are then of the shape:

$$\Gamma; \Sigma \vdash e:U.$$

Table 7 gives the typing rules, which are as expected. Most standard typing rules for boolean and arithmetic expressions are omitted. Note that we can derive the dual types for overlined names by rules $\lfloor \textsc{SessDual} \rfloor$ and $\lfloor \textsc{ChannDual} \rfloor$.

3.3 Typing rules for constraints

The judgements for constraints only say that they are well typed from a standard and a constraint session environments:

$$\Gamma; \Sigma \vdash C\ ok.$$

$$\frac{\Gamma;\, \Sigma_1 \vdash e_1 : U \quad \Gamma;\, \Sigma_2 \vdash e_2 : U \quad \Sigma_1 \asymp \Sigma_2}{\Gamma; \Sigma_1 \Cup \Sigma_2 \vdash e_1 \;=\; e_2 \; ok} \lfloor \textsc{Ok}\text{-}= \rfloor$$

$$\frac{\Gamma;\, \emptyset \vdash e_1 : \mathsf{Nat} \quad \Gamma;\, \emptyset \vdash e_2 : \mathsf{Nat}}{\Gamma; \emptyset \vdash e_1 \;\leq\; e_2 \; ok} \lfloor \textsc{Ok}\text{-}\leq \rfloor$$

$$\frac{\Gamma; \Sigma_1 \vdash C_1 \; ok \quad \Gamma; \Sigma_2 \vdash C_2 \; ok \quad \Sigma_1 \asymp \Sigma_2}{\Gamma; \Sigma_1 \Cup \Sigma_2 \vdash C_1 \mid C_2 \; ok} \lfloor \textsc{Conc}\text{-}\textsc{C} \rfloor$$

Table 8. Typing rules for constraints

The typing rules are given in Table 8. A single constraint is *ok* when it relates expressions of the same type (rules $\lfloor$Ok-=$\rfloor$ and $\lfloor$Ok-$\leq\rfloor$).

To get bilinearity of communication we want that each active channel has at most one dual active channel. We introduce an assumption $y : \bot$ to represent the fact that the channel name y, occurring in two constraints, has been used to prove duality between two other channel names (by means of rule [Trans]) but it cannot occur in pure processes. This mechanism is needed to correctly type channel delegation and process recursion. We must also guarantee that in putting together two expressions (rule $\lfloor$Ok-=$\rfloor$) or two sets of constraints (rule $\lfloor$Conc-C$\rfloor$) a channel name occurring twice has been used with the same type. This is assured by:

1. the condition that the constraint session environments of the premises agree, where *agreement* $\asymp$ between constraint session environments Σ and Σ' is given by:

$$\Sigma \asymp \Sigma' \text{ if } y \in dom(\Sigma) \cap dom(\Sigma') \text{ implies } \Sigma(y) = \Sigma'(y) \neq \bot;$$

2. taking as constraint session environment of the conclusion the *constraint union* $\Cup$ of the constraint session environments in the premises defined by:

$$\begin{aligned}\Sigma \Cup \Sigma' = \{y : \bot \mid y : T \in \Sigma \cap \Sigma'\} \cup \{y : T \mid y : T \in \Sigma \;\&\; y \notin dom(\Sigma')\} \cup \\ \{y : T \mid y : T \in \Sigma' \;\&\; y \notin dom(\Sigma)\}.\end{aligned}$$

In this way if $\Gamma; \Sigma \vdash C \; ok$ then:

- each channel name which occurs once in C gets a channel type in Σ;
- each channel name which occurs twice in C gets $\bot$ in Σ;
- no channel name occurs more than twice in C.

3.4 Typing rules for pure processes

In order to take into account the communications a pure process can offer, the typing judgements for pure processes derive finite mappings between channel names and channel types. More precisely we define a *pure session environment* Δ as a set of assumptions on channel names:

$$\Delta ::= \emptyset \mid \Delta, y : T$$

and then the judgements for pure processes are of the shape:

$$\Gamma \vdash P \triangleright \Delta.$$

$$\frac{\Gamma;\emptyset \vdash x : \langle T \rangle \quad \Gamma \vdash P \triangleright \Delta, y : T \quad \Gamma;\emptyset \vdash C \; ok}{\Gamma \vdash x\{C\}(y).P \triangleright \Delta} \lfloor \textsc{Sess} \rfloor \qquad \frac{\Gamma, x : B \vdash P \triangleright \Delta, y : T}{\Gamma, x : B \vdash y\langle\langle x \rangle\rangle.P \triangleright \Delta, y : B;T} \lfloor \textsc{Comm} \rfloor$$

$$\frac{\Gamma \vdash P \triangleright \Delta, y : T \quad z \notin dom(\Delta)}{\Gamma \vdash y\langle z \rangle.P \triangleright \Delta, y : !ch(T');T, z : T'} \lfloor \textsc{ChDel} \rfloor \qquad \frac{\Gamma \vdash P \triangleright \Delta, y : T, z : T'}{\Gamma \vdash y(z).P \triangleright \Delta, y : ?ch(T');T} \lfloor \textsc{ChRec} \rfloor$$

$$\frac{\Gamma \vdash P \triangleright \Delta, y : T_i \quad l_i : T_i \in \overrightarrow{l : T}}{\Gamma \vdash y \oplus l_i.P \triangleright \Delta, y : \oplus\{\overrightarrow{l : T}\}} \lfloor \textsc{Sel} \rfloor \qquad \frac{\Gamma \vdash P_i \triangleright \Delta, y : T_i \quad (1 \leq i \leq n)}{\Gamma \vdash y\&\{\overrightarrow{l : P}\} \triangleright \Delta, y : \&\{\overrightarrow{l : T}\}} \lfloor \textsc{Branch} \rfloor$$

$$\frac{\Gamma \vdash P \triangleright \Delta \qquad \Gamma;\emptyset \vdash \mathsf{c} \; ok}{\Gamma \vdash \mathtt{tell}(\mathsf{c});P \triangleright \Delta} \lfloor \textsc{Tell} \rfloor \qquad \frac{\Gamma \vdash P \triangleright \Delta \qquad \Gamma;\emptyset \vdash \mathsf{c} \; ok}{\Gamma \vdash \mathtt{check}(\mathsf{c}).P \triangleright \Delta} \lfloor \textsc{Check} \rfloor$$

$$\frac{\Gamma \vdash P_i \triangleright \Delta \quad \Gamma;\emptyset \vdash \mathsf{c}_i \; ok \quad (1 \leq i \leq n)}{\Gamma \vdash \mathtt{ask}(\overrightarrow{\mathsf{c} : P}) \triangleright \Delta} \lfloor \textsc{Ask} \rfloor \qquad \frac{\Gamma, x : B \vdash P \triangleright \Delta}{\Gamma \vdash (x)P \triangleright \Delta} \lfloor \textsc{Res} \rfloor$$

$$\frac{dom(\Gamma) \cap \overrightarrow{y} = \emptyset}{\Gamma \vdash \mathbf{0} \triangleright \overrightarrow{\{y : \mathsf{end}\}}} \lfloor \textsc{Inact} \rfloor \qquad \frac{\Gamma \vdash P \triangleright \Delta \qquad \Gamma \vdash Q \triangleright \Delta' \qquad dom(\Delta) \cap dom(\Delta') = \emptyset}{\Gamma \vdash P \mid Q \triangleright \Delta \cup \Delta'} \lfloor \textsc{Conc} \rfloor$$

$$\frac{\Gamma;\emptyset \vdash e : B}{\Gamma, X : BT \vdash X\langle e, z \rangle \triangleright \{z : T\}} \lfloor \textsc{Var} \rfloor \qquad \frac{\Gamma, X : BT, x : B \vdash P \triangleright \{y : T\} \qquad \Gamma, X : BT \vdash Q \triangleright \Delta}{\Gamma \vdash \mathsf{def}\, X(x,y) = P \;\mathsf{in}\; Q \triangleright \Delta} \lfloor \textsc{Def} \rfloor$$

Table 9. Typing rules for pure processes

Table 9 presents the typing rules for pure processes.

Rule $\lfloor$Sess$\rfloor$ allows to initialise a session which asks for the constraints C and uses the session name x and the channel name y. It requires that:

- the constraints C are *ok* and do not contain channel names;
- the channel type T of y is associated by the standard environment to x.

Note that the assumption $y : T$ is discharged in the pure session environment of the conclusion since the session initialisation restricts the name y.

A fusion is typed by rule $\lfloor$Comm$\rfloor$ which adds to the channel type T of y the sort type B of x.

After being delegated, a channel cannot be used any more by the delegating process: for this reason the channel z is added with its type T' to the pure session environment in the conclusion of rule $\lfloor$ChDel$\rfloor$. The reception of a channel z is a binding for z and therefore the assumption $z : T'$ is discharged in the conclusion of rule $\lfloor$ChRec$\rfloor$. In both rules $\lfloor$ChDel$\rfloor$ and $\lfloor$ChRec$\rfloor$ the type of the channel y is decorated with the information on the type of the exchanged channel.

Rules $\lfloor$Sel$\rfloor$ and $\lfloor$Branch$\rfloor$ associate to each label the type of the corresponding process.

Rules $\lfloor$Tell$\rfloor$, $\lfloor$Check$\rfloor$ and $\lfloor$Ask$\rfloor$ guarantee that the involved conditions are well typed. Rule $\lfloor$Ask$\rfloor$ requires that the different alternatives offer compatible sequence of communications representable with the same pure session environment.

$$\frac{\Gamma;\Sigma \vdash C\ ok}{\Gamma;\Sigma \vdash C \triangleright \emptyset}\ \lfloor \text{START-C} \rfloor \qquad \frac{\Gamma \vdash P \triangleright \Delta}{\Gamma;\emptyset \vdash P \triangleright \Delta}\ \lfloor \text{START-P} \rfloor$$

$$\frac{\Gamma;\Sigma \vdash S \triangleright \Delta \quad \Gamma;\Sigma' \vdash R \triangleright \Delta' \quad dom(\Delta) \cap dom(\Delta') = \emptyset \quad \Sigma \asymp \Sigma' \quad \Sigma \bowtie \Delta' \quad \Sigma' \bowtie \Delta}{\Gamma;\Sigma \uplus \Sigma' \vdash S \mid R \triangleright \Delta \cup \Delta'}\ \lfloor \text{CONC-S} \rfloor$$

$$\frac{\Gamma, x : B;\Sigma \vdash S \triangleright \Delta}{\Gamma;\Sigma \vdash (x)S \triangleright \Delta}\ \lfloor \text{RES-S} \rfloor \qquad \frac{\Gamma;\Sigma \vdash S \triangleright \Delta}{\Gamma;\Sigma \setminus y \vdash (y)S \triangleright \Delta \setminus y}\ \lfloor \text{RESCH} \rfloor$$

$$\frac{\Gamma, X : BT, x : B \vdash P \triangleright \{y : T\} \qquad \Gamma, X : BT;\Sigma \vdash S \triangleright \Delta}{\Gamma;\Sigma \vdash \mathsf{def}\ X(x,y) = P\ \mathsf{in}\ S \triangleright \Delta}\ \lfloor \text{DEF-S} \rfloor$$

Table 10. Typing rules for constrained processes

When putting two processes in parallel, to assure that the same channel does not occur in both, we put the condition $dom(\Delta) \cap dom(\Delta') = \emptyset$ as a premise of rule $\lfloor$CONC$\rfloor$.

Rule $\lfloor$INACT$\rfloor$ says that $\mathbf{0}$ cannot offer communications. The remaining rules are standard.

3.5 Typing rules for constrained processes

Constrained processes contain constraints in parallel with pure processes: so their typing requires both constraint and pure session environments. We have then for constrained processes judgements of the shape:

$$\Gamma;\Sigma \vdash S \triangleright \Delta.$$

Table 10 gives the rules to type constrained processes.

The start rules $\lfloor$START-C$\rfloor$ and $\lfloor$START-P$\rfloor$ promote constraints and pure processes to constrained processes, respectively.

The parallel of two constrained processes is well typed only if the constraint and pure session environments have crosswise the same assumptions for the common channels. This is checked by the conditions $\Sigma \bowtie \Delta'$ and $\Sigma' \bowtie \Delta$ in the premises of rule $\lfloor$CONC-S$\rfloor$, where we define

$$\Sigma \bowtie \Delta \text{ if for all } y \in dom(\Sigma) \cap dom(\Delta) \text{ we have } \Sigma(y) = \Delta(y).$$

In rule $\lfloor$RESCH$\rfloor$ $\Sigma \setminus y$ ($\Delta \setminus y$) is the environment Σ (Δ) in which an assumption on y, if any, has been erased.

For instance the bookseller example of Section 2.3 can be the typed in a standard environment containing $rate$: Real and the following assumptions for the session names a and b:

a: $\langle \mathsf{String}; \mathsf{Real}; \oplus\{\mathsf{ok} : \mathsf{String}; \mathsf{Nat}; \mathsf{end} \,[]\, \mathsf{quit} : \mathsf{end}\}\rangle$

b: $\langle \mathsf{Real}; \oplus\{\mathsf{ok} : !ch(\oplus\{\mathsf{ok} : \mathsf{String}; \mathsf{Nat}; \mathsf{end} \,[]\, \mathsf{quit} : \mathsf{end}\}); \mathsf{end} \,[]\, \mathsf{quit} : \mathsf{end}\}\rangle$

where we use $[]$ to separate the different alternatives in selection types. The type of b shows that the communication on a can be delegated after the fusions of one string and one real.

4 Main properties

A first basic and expected property is that structurally equivalent constrained processes have the same type, as stated in the following theorem.

Theorem 1. *Let $S \equiv R$ and $\Gamma; \Sigma \vdash S \triangleright \Delta$. Then $\Gamma : \Sigma \vdash R \triangleright \Delta$.*

As remarked before our type system assures bilinearity, i.e. that each active channel has at most one dual active channel. Moreover it assures *safety of communications*, in the sense that whenever an active channel and its dual channel are ready to communicate they offer communication actions which have the shapes required by rules [Comm], [Deleg] or [Branch], and the types of the data to be exchanged match. Therefore these actions can reduce, unless in rule [Comm] the fusion of data names would generate an inconsistent set of global constraints. An example of this fact is the well-typed process:

$$y_1 = \overline{y_2} \mid x_1 = 3 \mid x_2 = 5 \mid y_1\langle\langle x_1 \rangle\rangle.\mathbf{0} \mid y_2\langle\langle x_2 \rangle\rangle.\mathbf{0}.$$

In order to formalize the above properties we need some preliminary definitions. Let's denote by ϕ a communication action (of the form $y\langle\langle x \rangle\rangle$, $y\langle z \rangle$, $y(z)$, $y \oplus l$, or $y\&\{\overrightarrow{l : P}\}$). We say that a communication action ϕ is *ready* in a pure process P if $P \equiv P' \mid \phi.P''$ for some P', P''.

Let $\Gamma; \Sigma \vdash P \triangleright \Delta$. We say that a ready communication action ϕ in P *agrees* with a channel type T under Γ, Σ if we have that:

- $\phi = y\langle\langle x \rangle\rangle$ implies $T = B; T'$ and $\Gamma; \Sigma \vdash x : B$;
- $\phi = y\langle z \rangle$ implies $T = !ch(T''); T'$ and $\Gamma; \Sigma \vdash z : T''$;
- $\phi = y(z)$ implies $T = ?ch(T''); T'$;
- $\phi = y \oplus l$ implies $T = \oplus\{\overrightarrow{l : T}\}$ and $l \in \overrightarrow{l}$;
- $\phi = y\&\{\overrightarrow{l : P}\}$ implies $T = \&\{\overrightarrow{l : T}\}$.

Theorem 2 (Bilinearity and Communication Safety). *Let $\Gamma; \Sigma \vdash S \triangleright \Delta$ and $(\overrightarrow{\nu})(C \mid P)$ be the canonical form of S, then there are $\Gamma' \supseteq \Gamma$, $\Sigma' \supseteq \Sigma$, $\Delta' \supseteq \Delta$ such that $\Gamma'; \Sigma' \vdash C \mid P \triangleright \Delta'$ and:*

1. *for each channel name y free in P there are:*
   (a) *an assumption $y : T \in \Delta'$ for some channel type T;*
   (b) *at most one channel name z occurring free in P such that $C \vdash_c y = \overline{z}$ and in this case $z : \overline{T} \in \Delta'$ and $y : T, z : \overline{T} \in \Sigma'$;*
2. *if $y : T \in \Delta'$ and ϕ is a ready communication action with subject y in P, then ϕ agrees with T under Γ', Σ' and there cannot be any other ready communication action with subject y in P.*

As an immediate consequence of the above theorem if two communication actions with dual subjects y, z are both ready, then they can communicate (communication safety). Moreover no other ready communication action can have a subject which is dual of y or z (bilinearity).

Theorem 2 can be proved by induction on P by using the following facts:

1. typing allows at most two occurrences of a channel name in constraints,
2. Σ' and Δ' must give the same types to channel names which occur both in P and in C.

Our type system does not guarantee progress, i.e. that in a pure process with open sessions there is always at least a possible communication. The process given at the beginning of this section is well-typed and stuck. Another example is the process:

$$y_1 = \overline{y_2} \mid z_1 = \overline{z_2} \mid y_1\langle\langle x_1\rangle\rangle.z_1\langle\langle x_2\rangle\rangle.\mathbf{0} \mid z_2\langle\langle x_3\rangle\rangle.y_2\langle\langle x_4\rangle\rangle.\mathbf{0}$$

where the communications on the channels y and z are crossed. Note that the majority of type systems for session types do not assure progress, see [11] and the references there.

We state subject reduction, namely the invariance of typing judgements by reduction, under the hypothesis that the pure session environment is contained in the constraint session environment. This is not directly implied by the typing rules. A counter example is:

$$y_1 = \overline{y_2} \mid y_1\langle z_1\rangle.\mathbf{0} \mid y_2(z_2).\mathbf{0} \longrightarrow y_1 = \overline{y_2} \mid \mathbf{0} \mid (z_2)(z_1 = z_2 \mid \mathbf{0})$$

in which the lhs can be typed with Σ, Δ such that $dom(\Sigma) = \{y_1, y_2\}$ and $dom(\Delta) = \{y_1, y_2, z_1\}$, while the rhs requires an assumption for z_1 in the constraint session environment. This hypothesis is sensible since a channel can communicate only if the set of constraints proves that it is dual of another channel, see the reduction rules [Comm], [Deleg], and [Branch]. Note that one can easily prove that $\Gamma; \Sigma \vdash \overrightarrow{(v)}(C \mid P) \triangleright \Delta$ implies $fcn(\overrightarrow{(v)}C) = dom(\Sigma)$ and $fcn(\overrightarrow{(v)}P) \subseteq dom(\Delta)$, where $fcn(S)$ is the set of channel names which occur free in S. Then $\Delta \subseteq \Sigma$ guarantees also the bilinearity condition of the active channels, since all channels which occur twice in constraints get $\perp$ in Σ. It is easy to verify that the condition $\Delta \subseteq \Sigma$ is preserved by reduction.

As usual in calculi involving sessions (e.g. [5]) subject reduction does not hold literally, since channel types are shortened by reduction. The partial order between pairs of session environments defined next reflects the difference between channel types before and after one step reduction. Remark that the domain of constraint session environments does not change under reduction, while the domain of pure session environments may decrease.

Definition 1 (Evaluation Order).

1. *$\sqsubseteq$ is defined as the smallest partial order on channel types and $\perp$ such that:*
   $\perp \sqsubseteq T \qquad T \sqsubseteq B;T \qquad T \sqsubseteq \dagger ch(T');T \qquad T_i \sqsubseteq \oplus\overrightarrow{\{l : T\}} \qquad T_i \sqsubseteq \&\overrightarrow{\{l : T\}}.$
2. *$\sqsubseteq$ is extended to session environments as follows:*
   - *$\Sigma \sqsubseteq \Sigma'$ if $dom(\Sigma) = dom(\Sigma')$ and for all $y \in dom(\Sigma)$ we have $\Sigma(y) \sqsubseteq \Sigma'(y)$.*
   - *$\Delta \sqsubseteq \Delta'$ if for all $y \in dom(\Delta)$ we have $y \in dom(\Delta')$ and $\Delta(y) \sqsubseteq \Delta'(y)$.*

We are now able to state the subject reduction theorem.

Theorem 3 (Subject Reduction). *Let $\Gamma; \Sigma \vdash S \triangleright \Delta$ and $\Delta \subseteq \Sigma$. Then $S \longrightarrow S'$ imply $\Gamma; \Sigma' \vdash S' \triangleright \Delta'$ and $\Delta' \subseteq \Sigma'$ for some $\Sigma' \sqsubseteq \Sigma$, $\Delta' \sqsubseteq \Delta$.*

5 Conclusion and Future Works

This paper presents the first integration of sessions and concurrent constraints providing a calculus which allows to express both client/service requirements for opening an interaction and the interaction protocol, i.e. the sequence of communications which are expected to happen in the interaction itself.

We plan to develop our calculus in several different directions.

We will study the addition of a retract operator which allows to erase constraints [3], but we want to limit its use in such a way only the participants to a session can erase the constraints they put at the session start.

We would like to introduce a mechanism which makes the constraints required for opening a session local to the session itself.

It is interesting to enrich the type assignment system in order to assure progress inside sessions, i.e. that once a sessions has been initiated, processes will never starve at communication prefixes. The presence of constraints does not allow a simple adaptation of the tools developed for standard sessions [5].

Usability of our calculus will be enhanced by a type inference system which allows to automatically derive types for untyped constrained processes whenever possible.

References

1. S. Bistarelli, U. Montanari, and F. Rossi. Soft Concurrent Constraint Programming. *ACM Transactions on Computational Logic*, 7(3):1–27, 2006.
2. E. Bonelli, A. Compagnoni, and E. Gunter. Correspondence Assertions for Process Synchronization in Concurrent Communications. *Journal of Functional Programming*, 15(2):219–248, 2005.
3. M. Buscemi and U. Montanari. CC-Pi: A Constraint-Based Language for Specifying Service Level Agreements. In R. De Nicola, editor, *ESOP'07*, volume 4421 of *LNCS*, pages 18 –32. Springer, 2007.
4. M. Buscemi and U. Montanari. Open Bisimulation for the Concurrent Constraint Pi-Calculus. In S. Drossopoulou, editor, *ESOP'08*, volume 4960 of *LNCS*, pages 254–268. Springer, 2008.
5. M. Dezani-Ciancaglini, U. de' Liguoro, and N. Yoshida. On Progress for Structured Communications. In G. Barthe and C. Fournet, editors, *TGC'07*, volume 4912 of *LNCS*, pages 257–275, 2008.
6. K. Honda, V. T. Vasconcelos, and M. Kubo. Language Primitives and Type Disciplines for Structured Communication-based Programming. In C. Hankin, editor, *ESOP'98*, volume 1381 of *LNCS*, pages 22–138. Springer, 1998.
7. R. Milner, J. Parrow, and D. Walker. A Calculus of Mobile Processes, Parts I and II. *Information and Computation*, 100(1), 1992.
8. V. Saraswat and M. Rinard. Concurrent Constraint Programming. In F. E. Allen, editor, *POPL'90*, pages 232–245. ACM Press, 1990.
9. K. Takeuchi, K. Honda, and M. Kubo. An Interaction-based Language and its Typing System. In C. Halatsis, D. Maritsas, G. Philokyprou, and S. Theodoridis, editors, *PARLE'94*, volume 817 of *LNCS*, pages 398–413. Springer, 1994.
10. L. Wischik and P. Gardner. Explicit Fusion. *Theoretical Computer Science*, 340(3):606–630, 2005.
11. N. Yoshida and V. T. Vasconcelos. Language Primitives and Type Disciplines for Structured Communication-based Programming Revisited. In M. Fernández and C. Kirchner, editors, *SecRet'06*, volume 171 of *ENTCS*, pages 73–93. Elsevier, 2007.

A Appendix

A.1 Type preservation under structural equivalence

As usual we first prove that the well typing of an expression is invariant under structural equivalence. We start with some basic lemmas which can be easily shown by induction on deductions.

Lemma 1 (Weakening).

1. *If* $\Gamma;\Sigma \vdash C$ *ok and* $x \notin dom(\Gamma) \cup dom(\Sigma)$, *then* $\Gamma, x:B;\Sigma \vdash C$ *ok.*
2. *If* $\Gamma;\Sigma \vdash C$ *ok and* $X \notin dom(\Gamma)$, *then* $\Gamma, X:BT;\Sigma \vdash C$ *ok.*
3. *If* $\Gamma \vdash P \triangleright \Delta$ *and* $x \notin dom(\Gamma) \cup dom(\Delta)$, *then* $\Gamma, x:B \vdash P \triangleright \Delta$.
4. *If* $\Gamma \vdash P \triangleright \Delta$ *and* $X \notin dom(\Gamma)$, *then* $\Gamma, X:BT \vdash P \triangleright \Delta$.
5. *If* $\Gamma;\Sigma \vdash S \triangleright \Delta$ *and* $x \notin dom(\Gamma) \cup dom(\Sigma) \cup dom(\Delta)$, *then* $\Gamma, x:B;\Sigma \vdash S \triangleright \Delta$.
6. *If* $\Gamma;\Sigma \vdash S \triangleright \Delta$ *and* $X \notin dom(\Gamma)$, *then* $\Gamma, X:BT;\Sigma \vdash S \triangleright \Delta$.

Lemma 2 (Strengthening).

1. *If* $\Gamma, x:B;\Sigma \vdash C$ *ok and* $x \notin fn(C)$, *then* $\Gamma;\Sigma \vdash C$ *ok.*
2. *If* $\Gamma, X:BT;\Sigma \vdash C$ *ok, then* $\Gamma;\Sigma \vdash C$ *ok.*
3. *If* $\Gamma, x:B \vdash P \triangleright \Delta$ *and* $x \notin fn(P)$ *then* $\Gamma \vdash P \triangleright \Delta$.
4. *If* $\Gamma, X:BT \vdash P \triangleright \Delta$ *and* $X \notin fpv(P)$ *then* $\Gamma \vdash P \triangleright \Delta$.
5. *If* $\Gamma, x:B;\Sigma \vdash S \triangleright \Delta$ *and* $x \notin fn(S)$ *then* $\Gamma;\Sigma \vdash S \triangleright \Delta$.
6. *If* $\Gamma, X:BT;\Sigma \vdash S \triangleright \Delta$ *and* $X \notin fpv(S)$ *then* $\Gamma;\Sigma \vdash S \triangleright \Delta$.

Lemma 3. *Let* $\Gamma;\Sigma \vdash S \triangleright \Delta$, *then*

1. $dom(\Gamma) \cap dom(\Sigma) = dom(\Gamma) \cap dom(\Delta) = \emptyset$;
2. $dom(\Sigma) \cup dom(\Delta) \subseteq fn(S)$;
3. $\Sigma \bowtie \Delta$;
4. $y:\bot \in \Sigma$ *implies* $y \notin dom(\Delta)$.

Proof. By induction on derivations. Point (4) easily follows from (3).

Let $\sqsubseteq_0$ be the relation defined by $\bot \sqsubseteq_0 T$. We extend $\sqsubseteq_0$ to constraint session environments as in Definition 1. If Y is a set of channel names, we denote by $\Sigma \restriction Y$ the restriction of Σ to the names in Y, i.e. $\Sigma \restriction Y = \{y:T \in \Sigma \mid y \in Y\}$.

Lemma 4. *Let* $C \vdash_c c$ *and* $\Gamma;\Sigma \vdash C$ *ok. Then for some* Σ' *such that* $\Sigma \restriction fn(c) \sqsubseteq_0 \Sigma'$ *we have* $\Gamma;\Sigma' \vdash c$ *ok.*

The last lemma shows that $\asymp$ and $\Cup$ agree.

Lemma 5. 1. $\Sigma_1 \asymp \Sigma_2$ *and* $(\Sigma_1 \Cup \Sigma_2) \asymp \Sigma_3$ *iff* $\Sigma_1 \asymp (\Sigma_2 \Cup \Sigma_3)$ *and* $\Sigma_2 \asymp \Sigma_3$.
2. *If* $\Sigma_1 \asymp \Sigma_2$ *and* $(\Sigma_1 \Cup \Sigma_2) \asymp \Sigma_3$ *then* $(\Sigma_1 \Cup \Sigma_2) \Cup \Sigma_3 = \Sigma_1 \Cup (\Sigma_2 \Cup \Sigma_3)$.

Theorem 1 *Let* $S \equiv R$ *and* $\Gamma;\Sigma \vdash S \triangleright \Delta$. *Then* $\Gamma : \Sigma \vdash R \triangleright \Delta$.

Proof. By induction on the proof that $S \equiv R$. We discuss some interesting cases.

- *Case* $(\nu)S \mid R \equiv (\nu)(S \mid R)$ where $\nu \notin fn(R)$.

Note that, according to the fact that S and R are pure or constrained processes a typing for $(\nu)S \mid R$ can have been typed in different ways:

1. applying $\lfloor$RES$\rfloor$ to S and then $\lfloor$CONC$\rfloor$ to $(\nu)S$ and R if both S and R are pure processes. After that an application of rule $\lfloor$START-P$\rfloor$ could also be possible.
2. applying $\lfloor$RES$\rfloor$ and $\lfloor$START-P$\rfloor$to S and then rule $\lfloor$CONC-S$\rfloor$ if S is a pure process and R a constraint process.
3. Applying rules $\lfloor$RES-S$\rfloor$ or $\lfloor$RESCH$\rfloor$ to S and then rule $\lfloor$CONC-S$\rfloor$ if both S and R are constrained processes.

Let's consider case (3) in which the restriction has been obtained by rule $\lfloor$RESCH$\rfloor$.

$$\dfrac{\dfrac{\Gamma;\Sigma \vdash S \triangleright \Delta}{\Gamma;\Sigma' \vdash (y)S \triangleright \Delta'}\lfloor\text{RESCH}\rfloor \qquad \Gamma;\Sigma_1 \vdash R \triangleright \Delta_1 \quad dom(\Delta') \cap dom(\Delta_1) = \emptyset \quad \Sigma \asymp \Sigma_1 \quad \Sigma' \bowtie \Delta_1 \quad \Sigma_1 \bowtie \Delta'}{\Gamma;\Sigma' \uplus \Sigma_1 \vdash (y)S \mid R \triangleright \Delta' \cup \Delta_1}\lfloor\text{CONC-S}\rfloor$$

where $\Sigma' = \Sigma \setminus y$ and $\Delta' = \Delta \setminus y$. By Lemma 3(1) we have that y does not occur in Γ. Since $y \notin fn(R)$ by Lemma 3(2) we have that $y \notin dom(\Sigma_1)$ and $y \notin dom(\Delta_1)$. So $dom(\Delta) \cap dom(\Delta_1) = \emptyset$, $\Sigma \asymp \Sigma_1$ and also the involved $\bowtie$-relations are trivially preserved. So we can deduce:

$$\dfrac{\dfrac{\Gamma;\Sigma \vdash S \triangleright \Delta \quad \Gamma;\Sigma_1 \vdash R \triangleright \Delta_1 \quad dom(\Delta) \cap dom(\Delta_1) = \emptyset \quad \Sigma \asymp \Sigma_1 \quad \Sigma \bowtie \Delta_1 \quad \Sigma_1 \bowtie \Delta}{\Gamma;\Sigma \uplus \Sigma_1 \vdash S \mid R \triangleright \Delta \cup \Delta_1}\lfloor\text{CONC-S}\rfloor}{\Gamma;\Sigma \uplus \Sigma_1 \setminus y \vdash (y)(S \mid R) \triangleright \Delta \cup \Delta_1 \setminus y}\lfloor\text{RESCH}\rfloor$$

Since y does not occur in Σ_1, Δ_1 we have trivially that $\Sigma' \uplus \Sigma_1 = \Sigma \uplus \Sigma_1 \setminus y$ and $\Delta' \cup \Delta_1 = \Delta \cup \Delta_1 \setminus y$.

- *Case* $(S \mid R) \mid R' \equiv S \mid (R \mid R')$. A typing for $(S \mid R) \mid R'$ must have the form:

$$\dfrac{\dfrac{\Gamma;\Sigma_1 \vdash S \triangleright \Delta_1 \quad \Gamma;\Sigma_2 \vdash R \triangleright \Delta_2 \quad dom(\Delta_1) \cap dom(\Delta_2) = \emptyset \quad \Sigma_1 \asymp \Sigma_2 \quad \Sigma_1 \bowtie \Delta_2 \quad \Sigma_2 \bowtie \Delta_1}{\Gamma;\Sigma_1 \uplus \Sigma_2 \vdash (S \mid R) \triangleright \Delta_1 \cup \Delta_2}\lfloor\text{CONC-S}\rfloor \qquad \Gamma;\Sigma_3 \vdash R' \triangleright \Delta_3 \quad dom(\Delta_1 \cup \Delta_2) \cap dom(\Delta_3) = \emptyset \quad \Sigma_1 \uplus \Sigma_2 \asymp \Sigma_3 \quad \Sigma_1 \uplus \Sigma_2 \bowtie \Delta_3 \quad \Delta_1 \cup \Delta_2 \bowtie \Sigma_3}{\Gamma;(\Sigma_1 \uplus \Sigma_2) \uplus \Sigma_3 \vdash (S \mid R) \mid R' \triangleright \Delta_1 \cup \Delta_2 \cup \Delta_3}\lfloor\text{CONC-S}\rfloor$$

The condition on the application of the inference rules imply that $dom(\Delta_1)$, $dom(\Delta_2)$, $dom(\Delta_3)$ are disjoint.
Note that by Lemma 3(4), $\Sigma_1 \uplus \Sigma_2 \bowtie \Delta_3$ implies $\Sigma_2 \bowtie \Delta_3$. With similar arguments, and using Lemma 5(1), we have that the following is a valid deduction of the same type for $(S \mid R) \mid R'$:

$$\dfrac{\Gamma;\Sigma_1 \vdash S \triangleright \Delta_1 \qquad \dfrac{\Gamma;\Sigma_2 \vdash R \triangleright \Delta_2 \quad \Gamma;\Sigma_3 \vdash R' \triangleright \Delta_3 \quad dom(\Delta_2)\cap dom(\Delta_3)=\emptyset \quad \Sigma_2 \asymp \Sigma_3 \quad \Sigma_2 \bowtie \Delta_3 \quad \Sigma_3 \bowtie \Delta_2}{\Gamma;\Sigma_2 \Cup \Sigma_3 \vdash (R \mid R') \triangleright \Delta_2 \cup \Delta_3}\ \lfloor\textsc{Conc-S}\rfloor \quad dom(\Delta_2 \cup \Delta_3)\cap dom(\Delta_1)=\emptyset \quad \Sigma_1 \asymp \Sigma_2 \Cup \Sigma_3 \quad \Sigma_2 \Cup \Sigma_3 \bowtie \Delta_1 \quad \Sigma_1 \bowtie \Delta_2 \cup \Delta_3}{\Gamma;\Sigma_1 \Cup (\Sigma_2 \Cup \Sigma_3) \vdash S \mid (R \mid R') \triangleright \Delta_1 \cup \Delta_2 \cup \Delta_3}\ \lfloor\textsc{Conc-S}\rfloor$$

Lastly by Lemma 5(1) we have $\Sigma_1 \Cup (\Sigma_2 \Cup \Sigma_3) = (\Sigma_1 \Cup \Sigma_2) \Cup \Sigma_3$.

- *Case* $(\mathsf{def}\ D\ \mathsf{in}\ S) \mid R \equiv \mathsf{def}\ D\ \mathsf{in}\ (S \mid R)$ where $dpv(D) \notin fpv(R)$. A typing for $(\mathsf{def}\ D\ \mathsf{in}\ S) \mid R$ has the form:

$$\dfrac{\dfrac{\Gamma, X : BT, x : B \vdash P \triangleright y : T \qquad \Gamma, X : BT; \Sigma \vdash S \triangleright \Delta}{\Gamma;\Sigma \vdash \mathsf{def}\ X(x,y) = P\ \mathsf{in}\ S \triangleright \Delta}\ \lfloor\textsc{Def-S}\rfloor \qquad \Gamma;\Sigma' \vdash R \triangleright \Delta' \quad dom(\Delta)\cap dom(\Delta')=\emptyset \quad \Sigma \asymp \Sigma' \quad \Sigma \bowtie \Delta' \quad \Sigma' \bowtie \Delta}{\Gamma;\Sigma \Cup \Sigma' \vdash (\mathsf{def}\ X(x,y) = P\ \mathsf{in}\ S) \mid R \triangleright \Delta \cup \Delta'}\ \lfloor\textsc{Conc-S}\rfloor$$

Note that, since $X \notin fpv(R)$, by Lemma 1(6) and (possibly) (2) and (4), we have $\Gamma, X : BT; \Sigma' \vdash R \triangleright \Delta'$. By rule $\lfloor$Conc-S$\rfloor$ we can then prove $\Gamma, X : BT; \Sigma \Cup \Sigma' \vdash S \mid R \triangleright \Delta \cup \Delta'$ and then we conclude using rule $\lfloor$Def-S$\rfloor$.

A.2 Typing properties

A basic property of our constraints is the following, whose proof is straightforward by induction on derivations.

Lemma 6. *Let $\Gamma;\Sigma \vdash C$ ok. Then each channel name y can occur at most twice in C, and in this case $y : \bot \in \Sigma$.*

We say that a constraint c is *similar* to another constraint c' if c can be derived from c' using only rules $\lfloor$Axiom$\rfloor$, $\lfloor$Invol$\rfloor$, $\lfloor$Dual$\rfloor$, $\lfloor$Simm$\rfloor$. Note that this implies that c and c' contain the same names.

Lemma 7. *Let $C \vdash_c y = z$ ok and $C \vdash_c y = w$ ok, where z, w do not coincide. If each name occurs at most twice in C, then at least one between y, z, w occurs twice in C.*

Proof. By cases on the proof of $C \vdash_c y = z$ ok and $C \vdash_c y = w$ ok.

If C contains constraints similar to both $y = z$ and $y = w$, then trivially y occurs twice in C.

If C contains a constraint similar to $y = z$, but not to $y = w$, then C must contains constraints similar to $y = t_0$, $t_i = t_{i+1}$ and $t_n = w$ for $0 \le i \le n$, since a proof of $C \vdash_c y = w$ can only be obtained from constraints in C by applying the rules $\lfloor$Axiom$\rfloor$, $\lfloor$Invol$\rfloor$, $\lfloor$Dual$\rfloor$, $\lfloor$Simm$\rfloor$ and $\lfloor$Trans$\rfloor$. In this case if t_0, z coincide, then z occurs twice in C, and otherwise y occurs twice in C.

If C does not contain constraints similar to $y = z$ and to $y = w$, then C must contains constraints similar to $y = u_0$, $u_j = u_{j+1}$ and $u_m = z$ for $0 \le j \le m$, and to $y = t_0$, $t_i = t_{i+1}$ and $t_n = w$ for $0 \le i \le n$. Assume ad absurdum that u_i, t_i coincide for $0 \le i \le k$ but u_{k+1}, t_{k+1} do not coincide. This is impossible since u_k would appear three times in C. Then u_0, t_0 cannot coincide and y appears twice in C.

Theorem 2 (Bilinearity and Communication Safety). *Let $\Gamma;\Sigma \vdash S \triangleright \Delta$ and $\overrightarrow{(\nu)}(C \mid P)$ be the canonical form of S, then there are $\Gamma' \supseteq \Gamma$, $\Sigma' \supseteq \Sigma$, $\Delta' \supseteq \Delta$ such that $\Gamma';\Sigma' \vdash C \mid P \triangleright \Delta'$ and:*

1. *for each channel name y free in P there are:*
   (a) *an assumption $y : T \in \Delta'$ for some channel type T;*
   (b) *at most one channel name z occurring free in P such that $C \vdash_c y = \overline{z}$ and in this case $z : \overline{T} \in \Delta'$ and $y : T, z : \overline{T} \in \Sigma'$;*
2. *if $y : T \in \Delta'$ and ϕ is a ready communication action with subject y in P, then ϕ agrees with T under Γ', Σ' and there cannot be any other ready communication action with subject y in P.*

Proof. The restriction in $\overrightarrow{(\nu)}$ must have been typed by applications of $\lfloor$RES-S$\rfloor$ and $\lfloor$RESCH$\rfloor$ from a typing $\Gamma';\Sigma' \vdash C \mid P \triangleright \Delta'$. By Theorem 1 we can assume that the final steps of this deduction are the following:

$$\dfrac{\dfrac{\Gamma';\Sigma' \vdash C\ ok}{\Gamma', \Sigma' \vdash C \triangleright \emptyset}\ \lfloor \text{START-C} \rfloor \qquad \dfrac{\Gamma' \vdash P \triangleright \Delta'}{\Gamma';\emptyset \vdash P \triangleright \Delta'}\ \lfloor \text{START-P} \rfloor}{\Gamma';\Sigma' \vdash C \mid P \triangleright \Delta'}\ \lfloor \text{CONC-S} \rfloor$$

where we need only to assume $\Sigma' \bowtie \Delta'$.

The proof now is by induction on P. The basic step (P=**0**) is trivial. We give some interesting cases.

Case $P \equiv y\langle z' \rangle.P'$. Then the type deduction for P must end with:

$$\dfrac{\Gamma \vdash P' \triangleright \Delta'', y : T \quad z' \notin dom(\Delta')}{\Gamma \vdash y\langle z' \rangle.P' \triangleright \Delta'', y : !ch(T'); T, z' : T'}\ \lfloor \text{CHDEL} \rfloor$$

For point (1) by induction hypothesis for each channel name y_1 free in P' and different from y, z' there is an assumption $y_1 : T_1 \in \Delta'$ and there is at most one channel z_1 which occurs in P' such that $C \vdash_c y_1 = \overline{z_1}$ and in this case $z_1 : \overline{T_1} \in \Delta'$ and $y_1 : T_1, z_1 : \overline{T_1} \in \Sigma'$. By induction hypothesis we also have that z' does not occur in P' since $z' \notin dom(\Delta')$.

By Lemma 7 there is at most one channel z which occurs only once in C and such that $C \vdash_c y = \overline{z}$. Therefore the condition $\Sigma' \bowtie \Delta'$ implies that z is the only channel such that $C \vdash_c y = \overline{z}$ which can occur in P. From $\Gamma';\Sigma' \vdash C\ ok$ and Lemma 4 we get $\Sigma'(y) = \overline{\Sigma'(z)}$ and so we conclude $\Delta'(y) = \overline{\Delta'(z)}$ when z occurs in P again using $\Sigma' \bowtie \Delta'$.

The proof of point (2) is immediate.

Case $P \equiv P_1 \mid P_2$. Then the type deduction for P must end with:

$$\dfrac{\Gamma \vdash P_1 \triangleright \Delta_1 \quad \Gamma \vdash P_2 \triangleright \Delta_2 \quad dom(\Delta_1) \cap dom(\Delta_2) = \emptyset}{\Gamma \vdash P_1 \mid P_2 \triangleright \Delta_1 \cup \Delta_2}\ \lfloor \text{CONC} \rfloor$$

Point (1) follows easily by induction hypothesis. Note that by the condition $dom(\Delta_1) \cap dom(\Delta_2)$ a free channel name can occur either in Δ_1 or in Δ_2.
As for point (2) note that if ϕ is ready in $P_1 \mid P_2$ then ϕ must be ready in either P_1 or P_2. The proof follows immediately by induction.

A.3 Type preservation under reduction

Lemma 8. *If $\Gamma;\Sigma,y:T,z:\overline{T}\vdash C\ ok$ and $C\vdash_c y=\overline{z}$, then for any T' we get $\Gamma;\Sigma,y:T',z:\overline{T'}\vdash C\ ok$.*

Proof. Notice that by Lemma 6, y and z can occur only once in the constraints of C. If a constraint similar to $y=\overline{z}$ is a constraint of C, then we are done. Otherwise C must contains constraints similar to $y=w_0$, $w_i=w_{i+1}$ and $w_n=\overline{z}$ for $0\leq i\leq n$. Since all w_i occur twice in the showed constraints of C, by Lemma 6 they cannot occur in other constraints of C. This implies that $C\equiv C_1\mid C_2$, where C_1 only contains constraints similar to $y=w_0$, $w_i=w_{i+1}$ and $w_n=\overline{z}$ for $0\leq i\leq n$, while C_2 contains constraints without occurrences of $y,w_0,\ldots,w_n,\overline{z}$. Moreover $\Sigma=\Sigma',y:T,z:\overline{T},w_0:\bot,\ldots,w_n:\bot$ using again Lemma 6 and $\Gamma;y:T,z:\overline{T},w_0:\bot,\ldots,w_n:\bot\vdash C_1\ ok$ and $\Gamma;\Sigma'\vdash C_2\ ok$. It is then easy to verify that $\Gamma;y:T',z:\overline{T'},w_0:\bot,\ldots,w_n:\bot\vdash C_1\ ok$ for any T' and this implies the conclusion.

Theorem 3 (Subject Reduction). *Let $\Gamma;\Sigma\vdash S\triangleright\Delta$ where $\Delta\subseteq\Sigma$. Then $S\longrightarrow S'$ implies $\Gamma;\Sigma'\vdash S'\triangleright\Delta'$ and $\Delta'\subseteq\Sigma'$ for some $\Sigma'\sqsubseteq\Sigma$, $\Delta'\sqsubseteq\Delta$.*

Proof. By induction on $\longrightarrow$ and by cases. We only consider the most interesting cases.

- *Rule* [Link]:

$$C\mid x_1\{C_1\}(y_1)P_1\mid x_2\{C_2\}(y_2)P_2\ \longrightarrow\ C\mid C_1\mid C_2\mid(y_1)(y_2)(y_1=\overline{y_2}\mid P_1\mid P_2)$$

where $C\vdash_c x_1=\overline{x_2}$ and $cons(C\mid C_1\mid C_2)$ and $y_1\notin fn(P_2)$ and $y_2\notin fn(P_1)$. Since $\Gamma;\Sigma\vdash C\mid x_1\{C_1\}(y_1)P_1\mid x_2\{C_2\}(y_2)P_2\triangleright\Delta$ must be the conclusion of two applications of rule $\lfloor$CONC-S$\rfloor$ we get

$$\Gamma;\Sigma\vdash C\triangleright\emptyset \tag{1}$$
$$\Gamma;\emptyset\vdash x_1\{C_1\}(y_1)P_1\triangleright\Delta_1 \tag{2}$$
$$\Gamma;\emptyset\vdash x_2\{C_2\}(y_2)P_2\triangleright\Delta_2 \tag{3}$$

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

Since (1) must be the conclusion of rule $\lfloor$START-C$\rfloor$ we must have

$$\Gamma;\Sigma\vdash C\ ok \tag{4}$$

Since (2) must be the conclusion of rule $\lfloor$START-P$\rfloor$ applied to the conclusion of rule $\lfloor$SESS$\rfloor$ we have

$$\Gamma;\emptyset\vdash x_1:\langle T_1\rangle \tag{5}$$
$$\Gamma;\emptyset\vdash C_1\ ok \tag{6}$$
$$\Gamma\vdash P_1\triangleright\Delta_1,y_1:T_1 \tag{7}$$

and similarly from (3)

$$\Gamma;\emptyset\vdash x_2:\langle T_2\rangle \tag{8}$$
$$\Gamma;\emptyset\vdash C_2\ ok \tag{9}$$
$$\Gamma\vdash P_2\triangleright\Delta_2,y_2:T_2 \tag{10}$$

By applying rule $\lfloor$CONC-C$\rfloor$ to (4), (6), (9) we can derive $\Gamma;\Sigma \vdash C \mid C_1 \mid C_2\ ok$ and then by rule $\lfloor$START-C$\rfloor$

$$\Gamma;\Sigma \vdash C \mid C_1 \mid C_2 \triangleright \emptyset \tag{11}$$

Lemma 4 together with (4), (5), (8) and $C \vdash_c x_1 = \overline{x_2}$ imply $T_1 = \overline{T_2}$, which we will denote simply by T. By (7), (10) and rules $\lfloor$CHANN$\rfloor$, $\lfloor$CHANNDUAL$\rfloor$, $\lfloor$OK-=$\rfloor$ and $\lfloor$START-C$\rfloor$ we derive

$$\Gamma;\{y_1 : T, y_2 : \overline{T}\} \vdash y_1 = \overline{y_2} \triangleright \emptyset \tag{12}$$

Putting together (7), (10) and (12) using rule $\lfloor$CONC-S$\rfloor$ we get

$$\Gamma;\{y_1 : T, y_2 : \overline{T}\} \vdash y_1 = \overline{y_2} \mid P_1 \mid P_2 \triangleright \Delta_1 \cup \Delta_2, y_1 : T, y_2 : \overline{T}$$

and then using rule $\lfloor$RESCH$\rfloor$

$$\Gamma;\emptyset \vdash (y_1)(y_2)(y_1 = \overline{y_2} \mid P_1 \mid P_2) \triangleright \Delta_1 \cup \Delta_2 \tag{13}$$

We conclude by applying rule $\lfloor$CONC-S$\rfloor$ to (11) and (13).

- *Rule* [Deleg]:

$$C \mid y_1\langle z_1\rangle.P_1 \mid y_2(z_2).P_2 \ \longrightarrow\ C \mid P_1 \mid (z_2)(z_1 = z_2 \mid P_2)$$

where $C \vdash_c y_1 = \overline{y_2}$. As in previous case we must have:

$$\Gamma;\Sigma \vdash C\ ok \tag{14}$$
$$\Gamma \vdash y_1\langle z_1\rangle.P_1 \triangleright \Delta_1 \tag{15}$$
$$\Gamma \vdash y_2(z_2).P_2 \triangleright \Delta_2 \tag{16}$$

where $\Delta = \Delta_1 \cup \Delta_2$, $dom(\Delta_1) \cap dom(\Delta_2) = \emptyset$ and $\Sigma \bowtie \Delta_1 \cup \Delta_2$.

Since (15) must be the conclusion of rule $\lfloor$CHDEL$\rfloor$ we have

$$\Delta_1 = \Delta_1', y_1 : !ch(T_1'); T_1, z_1 : T_1' \quad \Gamma \vdash P_1 \triangleright \Delta_1', y_1 : T_1 \tag{17}$$

Similarly from (16) we must have

$$\Delta_2 = \Delta_2', y_2 : ?ch(T_2'); T_2 \quad \Gamma \vdash P_2 \triangleright \Delta_2', y_2 : T_2, z_2 : T_2' \tag{18}$$

Lemma 4 together with (17), (18) and $C \vdash_c y_1 = \overline{y_2}$ imply $T_1 = \overline{T_2}$, and $T_1' = T_2'$. We denote T_1, T_2 and T_1', T_2' by T and T', respectively.

Being $\Delta \subseteq \Sigma$ from (17) and (18) we must have

$$\Sigma = \Sigma'', y_1 : !ch(T'); T, y_2 : ?ch(T'); \overline{T}, z_1 : T'$$

and this by Lemma 8 and (14) gives $\Gamma;\Sigma'', y_1 : T, y_2 : \overline{T}, z_1 : T' \vdash C\ ok$, which implies by rule $\lfloor$START-C$\rfloor$

$$\Gamma;\Sigma'', y_1 : T, y_2 : \overline{T}, z_1 : T' \vdash C \triangleright \emptyset \tag{19}$$

By rules $\lfloor$CHANN$\rfloor$, $\lfloor$CHANNDUAL$\rfloor$, $\lfloor$OK-=$\rfloor$ and $\lfloor$START-C$\rfloor$ we derive

$$\Gamma;\{z_2 : T', z_1 : T'\} \vdash z_1 = z_2 \triangleright \emptyset$$

which together with (17) and (18) allows to get using rules $\lfloor$START-P$\rfloor$, $\lfloor$CONC-S$\rfloor$, and $\lfloor$RESCH$\rfloor$

$$\Gamma;\{z_1 : T'\} \vdash P_1 \mid (z_2)(z_1 = z_2 \mid P_2) \triangleright \Delta_1', \Delta_2', y_1 : T, y_2 : \overline{T} \tag{20}$$

By applying rule $\lfloor$CONC-S$\rfloor$ to (19) and (20) and observing that $\Sigma \bowtie \Delta_1 \cup \Delta_2$ implies $\Sigma'', y_1 : T, y_2 : \overline{T}, z_1 : \bot \bowtie \Delta_1', \Delta_2', y_1 : T, y_2 : \overline{T}$, since $z_1 \notin dom(\Delta_1') \cup dom(\Delta_2')$, we conclude:

$$\Gamma; \Sigma'', y_1 : T, y_2 : \overline{T}, z_1 : \bot \vdash C \mid P_1 \mid (z_2)(z_1 = z_2 \mid P_2) \triangleright \Delta_1', \Delta_2', y_1 : T, y_2 : \overline{T}.$$

- *Rule* [Rec]:

$$C \mid \mathsf{def}\, X(x,y) = P \;\mathsf{in}\; X(e,z) \;\longrightarrow\; C \mid \mathsf{def}\, X(x,y) = P \;\mathsf{in}\; (x)(y)(x{=}e \mid y{=}z \mid P)$$

As in previous case we get

$$\Gamma; \Sigma \vdash C \; ok \tag{21}$$

$$\Gamma \vdash \mathsf{def}\, X(x,y) = P \;\mathsf{in}\; X(e,z) \triangleright \Delta \tag{22}$$

Since (22) must be the conclusion of rule $\lfloor$DEF$\rfloor$ applied to the conclusion of rule $\lfloor$VAR$\rfloor$ we have $\Delta = \{z : T\}$ and $\Gamma, X : BT \vdash X(e,z) \triangleright \{z : T\}$ and

$$\Gamma, X : BT, x : B \vdash P \triangleright \{y : T\} \tag{23}$$

$$\Gamma; \emptyset \vdash e : B \tag{24}$$

By Lemma 1 from (24) we get $\Gamma, X : BT, x : B; \emptyset \vdash e : B$ and then using rules $\lfloor$ENV$\rfloor$ and $\lfloor$OK-=$\rfloor$

$$\Gamma, X : BT, x : B; \emptyset \vdash x = e \; ok \tag{25}$$

By rules $\lfloor$SENV$\rfloor$ and $\lfloor$OK-=$\rfloor$ we can also derive

$$\Gamma, X : BT, x : B; \{y : T, z : T\} \vdash y = z \; ok \tag{26}$$

From (23), (25), (26) by using rules $\lfloor$START-C$\rfloor$, $\lfloor$START-P$\rfloor$ and $\lfloor$CONC-S$\rfloor$ we get $\Gamma, X : BT, x : B; \{y : T, z : T\} \vdash x = e \mid y = z \mid P \triangleright \{y : T\}$ and then by rules $\lfloor$RES-S$\rfloor$ and $\lfloor$RESCH$\rfloor$

$$\Gamma, X : BT; \{z : T\} \vdash (x)(y)(x = e \mid y = z \mid P) \triangleright \emptyset \tag{27}$$

Applying rule $\lfloor$DEF-S$\rfloor$ to (23) and (27) we derive

$$\Gamma, X : BT; \{z : T\} \vdash \mathsf{def}\, X(x,y) = P \;\mathsf{in}\; (x)(y)(x = e \mid y = z \mid P) \triangleright \emptyset \tag{28}$$

Being $\{z : T\} \subseteq \Sigma$ by applying rule $\lfloor$START-C$\rfloor$ to (21) and then rule $\lfloor$CONC-S$\rfloor$ to the resulting judgement and to (28) we conclude

$$\Gamma, X : BT; \Sigma', z : \bot \vdash C \mid \mathsf{def}\, X(x,y) = P \;\mathsf{in}\; (x)(y)(x = e \mid y = z \mid P) \triangleright \emptyset$$

where $\Sigma = \Sigma', z : T$.