Correspondence Assertions for Process Synchronization in Concurrent Communications

Eduardo Bonelli\textsuperscript{1,3} Adriana Compagnoni\textsuperscript{1,4}

\textit{Stevens Institute of Technology}

Elsa Gunter\textsuperscript{1,2,5}

\textit{New Jersey Institute of Technology}

Abstract

High-level specification of patterns of communications such as protocols can be modeled elegantly by means of session types \cite{14}. However, a number of examples suggest that session types fall short when finer precision on protocol specification is required. In order to increase the expressiveness of session types we appeal to the theory of correspondence assertions \cite{6,10}. The resulting type discipline augments the types of long term channels with effects and thus yields types which may depend on messages read/written prior within the same session. We prove that evaluation preserves typability and that well-typed processes are safe. Also, we illustrate how the resulting theory allows us to address the shortcomings present in the pure theory of session types.

1 Introduction

Distributed and concurrent programming paradigms are increasingly popular, specially since the Internet entered the public domain. This has brought along new challenges including the specification and implementation of these programs together with techniques for the formal verification of their properties. One such specification method is that of protocol specification. This consists

\textsuperscript{1} This work was supported in part by the NSF Grant No. CCR-0220286 ITR:Secure Electronic Transactions.
\textsuperscript{2} It was also supported in part by the ARO under Award No. DAAD-19-01-1-0473
\textsuperscript{3} Email:ebonelli@cs.stevens-tech.edu
\textsuperscript{4} Email:abc@cs.stevens-tech.edu
\textsuperscript{5} Email:elsa@cis.njit.edu

\copyright 2003 Published by Elsevier Science B. V.
in identifying the sequence of message interchanges that take place between a number of parties in order to carry out some specific task. Recently, the use of type systems to formalize protocols has interested many researchers, in particular session types \[13,14\] has emerged as a promising approach. Interaction between a number of parties is achieved by specifying sequences of reciprocal interchanges of messages through private channels. Such sequences are modeled as types, the two parties at each end of the channel having dual such types. These pair of dual types constitute a session type. Session types are assigned to long term channels and are shared among processes. An example of a session type is:

\[\langle \downarrow \text{[Int]}; \downarrow \text{[Int]}; \uparrow \text{[Int]}; \uparrow \text{[Int]}; \downarrow \text{[Int]} \rangle\]

The first component indicates the expected behavior at one session point: the process must read an integer from the channel, then another one, and then write an integer to the channel (think of an “adding” server that reads in two numbers and writes out their sum). In order for the other party to interact correctly, it is assigned a dual type expression (the second component of the pair).

Quite some effort is being invested in the study of session types, motivated by the benefits that such a system provides for the analysis of protocols. Starting from the work of Honda et al \[13\], a suitable notion of subtypes for session types has been explored in \[12\], the benefits of session types in component based software development was presented in \[11\], bounded polymorphism in the presence of session types has been studied in \[12\], session types formulated in a \(\lambda\)-calculus with input/output operations is considered in \[13\].

This paper addresses a strengthening of session types by incorporating a theory of correspondence assertions (cf. Section \[12\]). We shall address a number of examples in which the shortcomings of session types are illustrated and shall exhibit how correspondence assertions successfully overcome these difficulties. The resulting type discipline is strictly richer than the pure theory of session types. More precisely, a number of “unsafe” programs which are well-typed in the theory of pure session types shall be rejected by our typing rules. To the best of our knowledge, this is the first study of a theory of correspondence assertions for long term channels.

1.1 Motivation

Consider the following example consisting of three parties: a Client, an ATM, and a Bank \[14\], as illustrated in Figure 1 which we briefly describe below:

**The Client.** A session is requested (through the shared name \(a\)), and then the Client sends its id number, selects a **deposit** operation, tells the amount of the deposit, and then waits for the new account balance.

**The ATM.** First it listens on name \(a\) for a client to request a session, then it reads in the client’s id number \((idA)\) and waits for the client’s selection
\[
C(id,\text{amt},a) = \text{request } a(k) \text{ in } k\text{!}[id]; \ k\leftarrow \text{deposit}; \ k\text{!}[\text{amt}]; \ k?(bal) \text{ in stop (1)}
\]

\[
\text{ATM}(a,b) = \text{accept } a(k) \text{ in } k?(idA) \text{ in}
\]
\[
k\triangleright \{ \text{deposit: request } b(h) \text{ in } k?(\text{amtA}) \text{ in}
\]
\[
\begin{array}{l}
\quad h\leftarrow \text{deposit}; \ h\text{!}[idA]; \ h\text{!}[\text{amtA}]; \ h?(\text{balA}) \text{ in } k\text{!}[\text{balA}]; \ \text{ATM}[a,b] \\
\quad \text{ withdrawal: request } b(h) \text{ in } k?(\text{amtA}) \text{ in}
\end{array}
\]
\[
\begin{array}{l}
\quad h\leftarrow \text{withdraw}; \ h\text{!}[idA]; \ h\text{!}[\text{amtA}]; h?(\text{OKedAmtA}) \text{ in } k\text{!}[\text{OKedAmtA}]; \ \text{ATM}[a,b] \}
\end{array}
\]

\[
\text{Bank}(b) = \text{accept } b(h) \text{ in } h?(idB) \text{ in } h?(\text{amtB}) \text{ in updateData; } h\text{!}[\text{balB}]; \ \text{Bank}[b] \text{ (2)}
\]

\[
\begin{array}{l}
\quad \text{ withdrawal: } h?(idB) \text{ in } h?(\text{amtB}) \text{ in}
\end{array}
\]
\[
\begin{array}{l}
\quad \text{getOK_AmtForIdB}; \ h\text{!}[\text{OKedAmtB}]; \ \text{Bank}[b] \}
\end{array}
\]

Fig. 1. The ATM example

of one of two available operations: deposit or withdraw. In the case of a deposit operation, the ATM requests a session with the bank (on name \(b\)), reads in the amount the client wishes to deposit (from \(a\)) and then selects the deposit operation of the Bank. It then sends the Bank the client’s id and the deposit amount, gets the new balance, reports it back to the client, and returns to the starting point. The ATM’s withdraw operation is similar.

The Bank. It listens on name \(b\) (shared with the ATM) for requests for a session, and then waits for the ATM to indicate the operation it wishes to perform (either deposit or withdraw). If it is a deposit operation, it reads in the id and the amount, updates its data, sends back the new balance, and then returns to its starting point. In the case of withdraw it proceeds accordingly.

Let the expression ATM|Client|Bank denote the concurrent execution of the indicated parties. The type system presented in [14] asserts that this expression is well-typed. Indeed, assigning the following session types to \(a\) and \(b\) (where \(σ(\alpha)\) is an abbreviation for the pair consisting of \(\alpha\) and its dual) we may type ATM|Client|Bank.

\[
a: σ(↓ [\text{Int}]; \&\{\text{deposit} : ↓ [\text{Int}]; ↑ [\text{Int}]; 1, \quad \text{ withdrawal} : ↓ [\text{Int}]; ↑ [\text{Int}]; 1})
\]
\[
b: σ(\&\{\text{deposit} : ↓ [\text{Int}]; ↓ [\text{Int}]; ↑ [\text{Int}]; 1, \quad \text{ withdrawal} : ↓ [\text{Int}]; ↓ [\text{Int}]; ↑ [\text{Int}]; 1})
\]

The first type says that all communication sessions established on \(a\) must abide to the communication pattern described by the argument of \(σ\) on one endpoint and its dual on the other. The latter argument type may be read as follows: after an integer is input, wait for one of two operations from the opposite endpoint deposit or withdraw, if deposit is selected then input an integer, output an integer and disallow further communication, likewise if
the operation selected is withdrawal. Note that these types express how the long term channels a and b behave independently of each other, even though they both belong to a common specification, namely that of the protocol specifying how Client, ATM, and Bank should interact in order to carry out a specific operation (a deposit or withdrawal). This may be witnessed as follows. Consider the ATM ATM’ resulting from ATM by replacing deposit with the following variant:

**Example 1.1** [Deposit I]

**deposit:**

\[
\begin{align*}
\text{request } & b(h) \text{ in } k?(amtA) \text{ in } h<\text{deposit}; h![idA]; h![amtA - 1.5]; \quad (3) \\
& h?(balA) \text{ in } k![balA]; \\
\text{request } & b(h') \text{ in } h'<\text{deposit}; h'![diffId]; h'![1.5]; h'? (balA') \quad (4) \\
\text{in } & \text{ATM}[a, b]
\end{align*}
\]

This version of the deposit operation deposits into the client’s account 1.5 units less than the amount told by the Client (3), and deposits the remaining 1.5 units in some account different from the client’s by means of a new deposit request (4) to the Bank which was not present in the original ATM.

Unfortunately, this modified ATM is typable under the same type assumptions as the previous one. Likewise, if the deposit operation of the good ATM were replaced by the same one except that the bank was not notified, then the resulting ATM also types under the same type assumptions as the good one.

**Example 1.2** [Deposit II] The following variant of deposit allows the ATM to keep the deposit of the Client without depositing it in the account. If we call the resulting system ATM", then ATM" | Client | Bank is well-typed under exactly the same type assumptions as ATM | Client | Bank.

**deposit:** k?(amtA) in k![1000]; ATM[a, b]

These examples suggest that although session types elegantly encode communication patterns of message interchange, they lack expressiveness in order to restrict interaction between sessions and also to enforce consistency of forwarded values (those received and then sent again). This paper introduces a type system based on correspondence assertions [21] in which ATM may be distinguished from the variants depicted in the above-mentioned examples.

### 1.2 Correspondence Assertions

Correspondence assertions originate in the context of model-checking [21]. In [3] a type system for correspondence assertions is presented for the spi-calculus; a neat presentation in the setting of an asynchronous \(\pi\)-calculus is
presented by the same authors in [14]. Roughly, they are used to formalize the idea that some point of execution in some process P must have been preceded by some other point of execution in some other process Q, in all possible executions of P|Q. Assertions are used to mark execution points in processes. As in [14], the assertions in this paper may have one of two forms: begin L or end L where L is an assertion label. A process is said to be safe if for every end L assertion reached in any execution, there is a corresponding begin L assertion which was reached sometime before in some other process.

By inserting appropriate correspondence assertions in suspicious code (including code communicating with the suspect code) and asking if the resulting code is safe, we may test for unexpected or malicious behavior. Safety may be determined by a type system hence allowing us to perform such checks statically.

Example 1.3 [Deposit I (cont)] Correspondence assertions allow us to show that the variant of ATM in Example 1.1 is unsafe if we assert that the amount to be deposited in the bank is the same as the amount given by the Client and appropriately augmenting the types of the sessions a and b. To show this, first we replace the code of Client by code including a begin assertion to obtain Client’:

request a(k) in begin ⟨id, amt⟩; k!⟨id⟩; k< deposit; k![amt]; k?(bal) in stop

Note that the label of the begin assertion contains an occurrence of the expressions id and amt. These are values generated by the Client and passed to the ATM. Next we add an end assertion to the deposit operation of Bank (2) in Figure 1 obtaining Bank’:

deposit: h?(idB) in h?(amtB) in end ⟨idB, amtB⟩;
updateData; h![balB]; Bank[b]

Finally, the session types of a and b are augmented with appropriate effects such that if the ATM receives an amount from the client, then it will be assigned a credit that it shall have to pay off by a corresponding communication with the bank.

Now the system ATM|Client’|Bank’ shall be safe if every time the Bank’s deposit operation is executed for an id number idB and amount amtB, the client requested the same operation on ATM, and idB = id, the id entered by the Client, and amtA = amtB, the amount entered by the Client. More importantly, ATM is forced to engage in communication with the bank, and moreover the deposit operation must be selected.

The type rules we present in this paper show that the system of Example 1.3 is unsafe for the given correspondence assertions. The question of how the type system forces the end assertion in Bank’ to be executed only after the corresponding begin assertion in Client’ has been executed is answered by means of latent effects on channels. In order to “reach” the end asser-
tion, the Bank’ must have previously executed the read operations of deposit (i.e. \( h\overline{2}(idB) \) in \( h\overline{2}(amtB) \)). Now, \( h \) is a channel which is shared between Bank’ and ATM'|Client’ (via ATM’). Via the placement of latent effects on the channel \( h \), Bank’ may pass back to whomever tries to send values on that channel the obligation of matching the end assertion. Similarly, ATM’ can use latent effects on the channel it shares with the Client to further pass along the obligation. In fact, since the ATM’ code has no assertions of it’s own, that is all it can do with the obligation. As the obligation is passed back through latent effects, it must be translated with respect to the substitution taking place as a result of the message passing on the channel. As the obligation is passed back from the Bank’ to the ATM’, it becomes \( \langle idA, amtA - 1.5 \rangle \), since these are the amounts sent for idB and amtB. As we pass the obligation back to Client, it is further transformed to \( \langle id, amt - 1.5 \rangle \), which we see does not match with the assertion begin \( \langle id, amt \rangle \). We may conclude, therefore, that the program is not safe. It is worth noting that if we changed the begin assertion to begin \( \langle id, amt - 1.5 \rangle \), then the program would type check and be declared safe. We would, in effect, be acknowledging that ATM’ had a right to charge a 1.5 unit fee for a deposit transaction.

**Contribution.** In this paper we introduce a type based theory of correspondence assertions for session types.

- In contrast to previous type systems for such assertions, session types allow the effects of an input/output type to depend on messages which were interchanged prior in the same session. We also include the branching/selection and delegation constructs from [14] in our analysis. The resulting type system shall allow us to distinguish the three above-mentioned variants of the ATM. This is achieved by introducing appropriate type directives (i.e. assertions) in the code and assigning appropriate types to names and channels; then type checking using the type discipline presented in this paper.

- The presence of a dependent types in a setting where types for long term channels are present introduces a number of technical difficulties. For example, the usual representation of environments as sequences of assumptions [210] fails to yield a calculus with basic properties such as admissibility of structural rules (cf. Remark 2.3). Also, recording of effects in closed channels is crucial in order to benefit from properties such as Subject Congruence (cf. Section 2.2.1 and 3).

- We show that evaluation preserves typability and that processes typable under empty effects are safe.

**Related work.** This work may be included among others in which type systems for the \( \pi \)-calculus are studied [15,17,16,20]. More closely to the present work [12] introduces subtyping into session types, however the concept of synchronization between sessions is not explored. The same holds for [22] and [19], the first studies a typing scheme for processes based on graph types and the second a type system for restricting communication in concurrent objects;
their relation to session types is discussed in \cite{14}. While \cite{10} shares a fair amount in common with this work, there is a major difference. In \cite{10} dependencies in types are “horizontal” in the sense that in a type expression such as $\downarrow [x : T_1, y : T_2]$ the type of $y$ may depend on the value of $x$, this being fixed for all communications over a channel of this type. However, since our setting is that of session types we allow “vertical” dependencies of the form $\downarrow [x : T_1];&\{l_1 : \downarrow [y : T_2], l_2 : \downarrow [z : T_3];&\downarrow [y : T_2]\}$. In this case, after reading a value for $x$, depending on the branching label selected either the type of $y$ may depend on $x$ (if label $l_1$ was selected) or the type of $y$ may depend on $x$ and $z$ (if the label $l_2$ was selected). Thus, in the present work, dependency spans whole sessions. Recently, type systems where CCS-like processes are used for typing process expressions have appeared. The generic type systems of \cite{15} is an example, although they do not incorporate correspondence assertions (however see Section \ref{sec:4}). Another approach is \cite{5} in which models (types as CCS-processes) of $\pi$-calculus expressions are obtained and the validity of temporal formulas are analyzed through model-checking techniques in order to deduce properties of the process expressions. They propose a type-and-effect system which incorporates correspondence assertions, however no long term channel types are available.

**Structure of the paper.** Section \ref{sec:2} adapts the calculus of \cite{14} to our setting (which we call $\pi_s$) and extends it with correspondence assertions. Section \ref{sec:2.2.1} presents a type system with effects for $\pi_s$. The proof of safety is given in Section \ref{sec:3} by introducing an appropriate labeled transition semantics. Finally we conclude and suggest further research directions.

2 The $\pi_s$-Calculus

2.1 Syntax

This section describes the syntax of $\pi_s$. We begin with a set of names $x, y, z, \ldots$. Within these names we distinguish: $a, b, \ldots$ for session names, $k, k', \ldots$ for channel names, $x, y, z, \ldots$ for variables. We also have integer constants $\ldots, -1, 0, 1, \ldots$, (branching) labels $l, l', \ldots$ and process variables written $X, Y, \ldots$. Process expressions, denoted with $P, Q, \ldots$, are defined as follows:

\[
P ::= \text{request } a(k) \text{ in } P \mid \text{accept } a(k) \text{ in } P \mid k? (x) \text{ in } P \mid k! [v]; P \mid k < l; P \mid k > \{l_1 : P_1 \square \ldots \square l_n : P_n\} \mid \text{throw } k[k']; P \mid \text{catch } k(k') \text{ in } P \mid P \mid Q \mid \text{stop} \mid (\nu a : T) P \mid (\nu k) P \mid \text{def } D \text{ in } P \mid X[\bar{v}] \mid \text{begin } L; P \mid \text{end } L; P
\]

*Process definitions* $D$ take the form $X_1[\bar{x_1}] = P_1$ and $\ldots$ and $X_n[\bar{x_n}] = P_n$.

**Remark 2.1** Parenthesis are binding constructs. Any two process expressions which differ only in the names of their bound names (called $\alpha$-equivalent)
shall be considered equal. A value is a name or an integer constant and is
denoted with letter \( v, v' \), \ldots. We use the notation \( P\{x \leftarrow v\} \) for the result of
substituting all free occurrences of \( x \) in \( P \) by \( v \). Note that for the benefit of a
clear presentation we have chosen to present a monadic calculus; an extension
to the polyadic case should be straightforward.

The request primitive requests a session on name \( a \). When this session is
established the fresh private channel \( k \) shall be used for message interchange.
The accept receives a request on the same name \( a \) and generates a new private
channel for message interchange to be used once the session is established.
The request and accept constructs each bind all free occurrences of the \( i \)
immediately following channel variable, \( k \), in the subsequent process, \( P \). The
synchronous sending and receiving of messages is achieved with \( k!v]Q \) and
\( k?x\) in \( P \) respectively, although, as in \([14]\), a translation to an asynchronous
calculus with branching is possible. A mechanism for selection of a label and
branching is available as \( k < l \) in \( P \) and \( k > \{ l_1 : P_1 \sqcap \ldots \sqcap l_n : P_n \} \); controlled
side-stepping of linearity constraints on channel usage is achieved by means of
channel delegation \( \text{throw } k[k'] \) in \( P \) and \( \text{catch } k(k') \) in \( Q \). The notation \( P \mid Q \)
has already been explained, also we use \text{stop} for inaction. We write \( (\nu v)P \)
for \( (\nu v : T)P \) or \( (\nu v)P \), the usual constructs for name hiding; \( T \) denotes
a type expression (Definition \([2.2]\)). Definitions of processes are also allowed
through the \text{def } D \text{ in } P \text{ construct, possibly introducing recursion. The begin
and end assertions shall be used as type directives in the type system for } \pi_s,
(Section \([2.2.1]\)): \text{begin } \langle v_1, \ldots, v_n \rangle \text{;} \text{P simply asserts } \text{begin } L \text{ and then behaves as } \text{P},
likewise \text{end } \langle v_1, \ldots, v_n \rangle \text{;} \text{P asserts } \text{end } L \text{ and then behaves as } \text{P}. Assertion labels are
tuples of values written \( \langle v_1, \ldots, v_n \rangle \).

2.2 The Type Discipline

The present section enriches the type system of \([14]\) with correspondence asser-
tions in order to address the shortcomings mentioned in the introduction.

2.2.1 Session types and effects:
The type system shall assign an effect to a process under some set of type
assumptions. The effect of a process reflects the pending obligations it has.
An assertion of the form \text{begin } L \text{ shall reduce these obligations by withdrawing
the assertion label } L \text{ from the current effect; likewise } \text{end } L \text{ shall augment the
current effect with } L \text{. Thus effects determine upper-bounds as to the number of
begin assertions that may be present. If the process has as empty effect,
then all } \text{end } \text{ assertions correspond to a matching } \text{begin } \text{ assertion.}

As explained above, effects also have to be attached to channel types in
order for two or more processes to share information on their pending or latent
effects. Effects added to channels are thus called latent effects.

\textbf{Definition 2.2} [Types with Effects] Assertion labels, effects and types are
given by the following grammar:

\[
\begin{align*}
\text{Plain Type} & \quad T \ ::= \ \text{Int} \ | \ \sigma(\alpha) \\
\text{Channel type} & \quad \alpha, \beta \ ::= \downarrow [x : T]e\alpha \ | \ \uparrow [x : T]e\alpha \ | \ \downarrow [k : \alpha]e\beta \\
& \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \ | \ \uparrow [k : \alpha]e\beta \ | \ \&\{l_1 : \alpha_1, \ldots, l_n : \alpha_n\}e \\
& \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \ | \ \oplus\{l_1 : \alpha_1, \ldots, l_n : \alpha_n\}e \\
\text{Effect} & \quad e, e' \ ::= \{L_1, \ldots, L_n\} \\
\text{Assertion Label} & \quad L, L_i \ ::= \langle v_1, \ldots, v_n\rangle
\end{align*}
\]

A type is either a plain type or a channel type; we use \(U, U_i\) to range over types. The set of free names of a type \(U\), written \(\text{fn}(U)\), is defined as usual (see [3]). The base type \(\text{Int}\) is the type of integer constants. Session types are represented as \(\sigma(\alpha)\) and may informally be seen to denote a pair consisting of a channel type \(\alpha\) and its dual \(\overline{\alpha}\):

\[
\begin{align*}
\downarrow [U]e\alpha & \overset{\text{def}}{=} \uparrow [U];\overline{\alpha} & \&\{l_i : \alpha_i\}e & \overset{\text{def}}{=} \oplus\{l_i : \alpha_i\}e & \overline{1} & \overset{\text{def}}{=} 1 \\
\downarrow [U]e\alpha & \overset{\text{def}}{=} \uparrow [U];\overline{\alpha} & \oplus\{l_i : \alpha_i\}e & \overset{\text{def}}{=} \&\{l_i : \alpha_i\}e
\end{align*}
\]

The types \(\alpha\) and \(\overline{\alpha}\) shall be assigned to the two endpoints of a communication session. A channel type consists of a sequence of input/output types of values or channels, or branch/selection types; the sequence is assumed to terminate with the channel type terminator \(1\). Each of these is accompanied by a latent effect. An effect is a multi-set of assertion labels; we use \(\{\ldots\}\) for the multi-set constructor. The special channel type \(\perp_e\) models a “complete” or “closed” channel which is already being used by two existing endpoints and thus through which no further communication is possible (cf. Definition [2.4]).

### 2.2.2 Typing Rules:

An environment \(\Gamma\) is a set of type assumptions \(x_1 : U_1, \ldots, x_n : U_n\) where \(x_1, \ldots, x_n\) are distinct names. We use letters \(\Gamma, \Delta, \ldots\) for environments. The domain of \(\Gamma\), written \(\text{dom}(\Gamma)\), is the set \(\{x_1, \ldots, x_n\}\). Also, we write \(\text{domCh}(\Gamma)\) for the subset of names to which \(\Gamma\) assigns channel types and \(\text{domPl}(\Gamma)\) for the subset of names to which \(\Gamma\) assigns plain types. In an assumption \(x : U\), \(x\) is called the subject; if the type assigned to the subject is a plain type then the assumption is said to be a plain assumption, otherwise it is a channel assumption. The notation \(\Gamma \setminus x : U\) stands for the environment resulting from dropping the assumption \(x : U\) from \(\Gamma\) (assuming it exists). In order for an environment to be well-formed it must obey the following two conditions:

**C1.** For each \(i \in 1..n\), \(\text{fn}(U_i) \subseteq \text{domPl}(\Gamma) \setminus \{x_i\}\)

**C2.** Define the relation \(x_i : U_i\) depends on \(x_j : U_j\) in \(\Gamma\) (written \((x_j : U_j) \rightarrow (x_i : U_i)\)) to hold if \(x_j \in \text{fn}(U_i)\). We require that the transitive closure of \(\rightarrow\) is irreflexive.
Condition C1 combines two requirements in one: the first is that all free names in types assigned by $\Gamma$ must be declared within $\Gamma$, moreover these types may only depend on names which are assigned plain types. Since interaction through channel names is restricted by linearity conditions (they are linear assumptions), this condition states that we do not allow types depending on linear assumptions (we do however allow types depending on plain or “intuitionistic” assumptions). The intended application of our type discipline is not disturbed by such a restriction and it is not clear whether the technical complications of the meta-theory resulting from lifting it outweighs its benefits. In fact this restriction appears in other settings in which linear and intuitionistic assumptions live together such as the linear logical framework of [4]. The second condition, C2, requires that $\Gamma$ have no cyclic dependencies. This is usually guaranteed by the representation of environments as sequences of type assumptions, in which an assumption $x : U$ depends only on those appearing to its left. Such a representation seems unfit in a setting where channel types are present since basic results on admissibility of structural rules fail (Rem. 2.5).

The judgments of the type discipline proposed for $\pi_s$ may take one of the following forms, where “wf” abbreviates “well-formed”, and are defined using four sets of rules:

$$
\begin{align*}
\Gamma \vdash \Theta \quad & \text{wf environment } \Gamma \text{ and process protocol } \Theta \\
\Gamma \vdash v : U \quad & \text{wf value } v \text{ of type } U \\
\Gamma \vdash (\vec{v}) : (\vec{x} : \vec{U}) \quad & \text{wf process parameters } \vec{v} \text{ of type } (\vec{x} : \vec{U}) \\
\Gamma \vdash P : e \quad & \text{wf process } P \text{ with effect } e
\end{align*}
$$

The letter $\Theta$ stands for a process protocol: a set of assumptions on the types of process parameters $X_j : (\vec{x}_j : \vec{U}_j)$ where each $\vec{x}_j : \vec{U}_j$ is a set of type assumptions for process $X_j$. Process protocols are subject to the same conditions of well-formedness as environments are: each $\vec{x}_j : \vec{U}_j$ must satisfy conditions C1 and C2.

The type rules of $\pi_s$ are presented in Figure 2. The rules Type Acpt and Type Rcv introduce a new channel name in the environment thus guaranteeing that a private channel is being used for the session. Note that dual channel types are used for the requesting and accepting parties. Type Bgn and Type End affect process effects by eliminating or adding a new assertion label (“+” stands for multiset union). The rules Type Snd and Type Rcv allow the typing of the communication primitives for sending and receiving data. Note that data is sent and received over channels only. Also, note that the type of $k$ in the upper righthand judgment of Type Snd is $\alpha\{y \leftarrow v\}$ reflecting the fact that the “rest” of the channel type, namely $\alpha$, may depend on the output value $v$. The same comment applies to the Type Rcv rule although this time, since both occurrences of $y$ (the one in $\downarrow [y : T]e'$; $\alpha$ and $k?y(y) \text{ in } P$) are bound we may assume by $\alpha$-conversion that they coincide. Type Brnch and
Type Sel type the branching and selection primitives, respectively; if pending effects are seen as credits, then it is clear that the effects of each branch in Type Branch be joined. Channel delegation is achieved by means of the throw and catch primitives which are typed by means of Type Thr and Type Cat. The rule Type Thr is subject to the restriction that $\beta \neq 1$; this restricts delegation of channels to those through which communication is possible i.e. no “dead” channel. Channel and name restriction (for non-channel names) are typed as expected. The Type Subsum rule allows to increase the required assertion obligations of a process term. The Type Par rule types the parallel execution of two processes. Since channels usage must be restricted in order to guarantee linear usage the environments $\Gamma$ and $\Gamma'$ are required to be compatible.

Definition 2.3 [Compatibility $\bowtie$] The relation $\bowtie$ is defined as follows: $\emptyset \bowtie \emptyset$, and $\Gamma \bowtie \Gamma'$ implies

(i) $\Gamma \cdot x : T \bowtie \Gamma' \cdot x : T$
(ii) $\Gamma \cdot k : \alpha \bowtie \Gamma' \cdot k : \alpha$
(iii) $\Gamma \cdot k : \alpha \bowtie \Gamma'$, if $k \notin \text{dom}(\Gamma')$
(iv) $\Gamma \bowtie \Gamma' \cdot k : \alpha$, if $k \notin \text{dom}(\Gamma)$

Note that the notion of compatibility makes sense for two sets of assumptions which not necessarily constitute well-formed environments. Once this notion of compatibility is in place we may define how two environments are combined through environment composition.

Definition 2.4 [Composition $\circ$] Let $\Gamma$, $\Gamma'$ be two environments such that $\Gamma \bowtie \Gamma'$. We define $\Gamma \circ \Gamma'$ as follows: $\emptyset \circ \emptyset = \emptyset$ and

(i) $(\Gamma \cdot x : T) \circ (\Gamma' \cdot x : T) = (\Gamma \circ \Gamma') \cdot x : T$
(ii) $(\Gamma \cdot k : \alpha) \circ (\Gamma' \cdot k : \alpha) = (\Gamma \circ \Gamma') \cdot k : \bot_{\text{fnMult}}(\alpha)$
(iii) $(\Gamma \cdot k : \alpha) \circ (\Gamma') = (\Gamma \circ \Gamma') \cdot k : \alpha$, if $k \notin \text{dom}(\Gamma')$
(iv) $\Gamma \circ (\Gamma' \cdot k : \alpha) = (\Gamma \circ \Gamma') \cdot k : \alpha$, if $k \notin \text{dom}(\Gamma)$

The effect $\text{fnMult}(\alpha)$ is the multiset which includes a label for each occurrence of a free name in $\alpha$. Other variants for the second clause of Definition 2.4 are possible as long as the effect subscript of $\bot$ faithfully records the name dependencies of the dual channel types from which it arises (i.e. no dependency information is lost).

For the sake of readability, in Figure 2 we have ommitted the hypothesis that the environment of the conclusion of the rule be well-formed, for all those rules where the environment of the conclusion is different from the environment of all hypothesis. Note that in some of the latter rules the condition is superfluous, namely Type CRes and Type Par.

---

6 Technically, this allows us to correct a problem present in [14], namely the failure of Subject Congruence.
\[
\begin{align*}
\Gamma \cdot a : \sigma(\alpha) \cdot k : \alpha \vdash_\Theta P : e & \quad \text{Type Acpt} \\
\Gamma \cdot a : \sigma(\alpha) \vdash_\Theta \text{accept } a(k) \text{ in } P : e & \\
\Gamma \cdot a : \sigma(\alpha) \cdot k : \pi \vdash_\Theta P : e & \quad \text{Type Requ} \\
\Gamma \cdot a : \sigma(\alpha) \vdash_\Theta \text{request } a(k) \text{ in } P : e & \\
\Gamma \vdash_\Theta P : e \quad \text{fn}(L) \subseteq \text{dom}(\Gamma) & \quad \text{Type Bgn} \\
\Gamma \vdash_\Theta \text{begin } L ; P : e - \{ L \} & \\
\Gamma \vdash_\Theta \text{end } L ; P : e + \{ L \} & \quad \text{Type End} \\
\Gamma \vdash_\Theta \text{begin } L ; P : e - y & \\
\Gamma \vdash_\Theta \text{end } L ; P : e + y & \\
\Gamma \vdash_\Theta k : y \vdash_\Theta P : e & \\
\Gamma \vdash_\Theta k \vdash_\Theta \text{throw } k[y] ; P : e + y & \quad \text{Type Thr} \\
\Gamma \vdash_\Theta k : y \vdash_\Theta \text{catch } k[y] \text{ in } P : e - y & \\
\Gamma \vdash_\Theta k : \alpha \vdash_\Theta P : e & \\
\Gamma \vdash_\Theta \alpha \vdash_\Theta \text{runCh}(\Gamma) \subseteq \{ 1, \bot \} & \quad \text{Type Stop} \\
\Gamma \vdash_\Theta \alpha \vdash_\Theta P : e & \\
\Gamma \vdash_\Theta \alpha \vdash_\Theta (\nu k) P : e & \quad \text{Type CRRes} \\
\Gamma \vdash_\Theta P : e \quad \Gamma \vdash_\Theta Q : e' & \quad \Gamma \vdash_\Theta \Gamma' \\
\Gamma \vdash_\Theta \text{par } \Gamma \vdash_\Theta P \vdash_\Theta Q : e + e' & \quad \text{Type Par} \\
\Gamma \vdash_\Theta P : e \quad e \leq e' \quad \text{fn}(e') \subseteq \text{dom}(\Gamma) & \quad \text{Type Subsum}
\end{align*}
\]

Fig. 2. Well-formed process expressions

**Remark 2.5** A representation of environments based on sequences of hypothesis, as usually adopted in the literature on dependent type systems [2], is not applicable to our system. The reason is that basic results on the admissibility of structural rules fail. In particular, the Exchange Lemma, which states that the order of independent hypothesis is irrelevant for the sake of derivability
\[
\Gamma \vdash_\Theta (\bar{v} : (\bar{x} : \bar{T})) \quad X : (\bar{x} : \bar{T}) \in \Theta \quad \text{ranCh}(\Gamma) \subseteq \{1, \perp\} \quad \text{Type PVar}
\]
\[
\Gamma \vdash_\Theta X[\bar{v}] : (\perp)'
\]
\[
\Gamma \setminus \text{chan}(\Gamma) \cdot x : \bar{x}_i : \bar{T}_i \vdash_\Theta P_i : (\perp) \quad \text{fn}(\bar{T}_i) \cap \text{dom}(\Gamma) = \emptyset
\]
\[
\Theta(X_i) = (\bar{x}_i : \bar{T}_i) \quad \Gamma \vdash_\Theta Q : e
\]
\[
\Gamma \vdash_\Theta X \text{ def } X_1(\bar{x}_1) = P_1 \ldots \text{ and } \ldots X_n(\bar{x}_n) = P_n \text{ in } Q : e
\]

Fig. 3. Well-formed process variables and definitions

\[
\begin{align*}
\Gamma \cdot x : U \cdot \Gamma' & \vdash_\Theta \circ & \text{Wf Val Name} \\
\Gamma \cdot x : U \cdot \Gamma' & \vdash_\Theta x : U & \text{Wf Val Int}
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash_\Theta \circ & \text{Wf PP Nil} \\
\Gamma \vdash_\Theta () : () & \text{Wf PP Cons}
\end{align*}
\]

Fig. 4. Well-formed values and process parameters

fails. Indeed, consider the following possible type rule Type Snd formulated in
a setting where environments are sequences:
\[
\Gamma_1 \cdot \Gamma_2 \vdash_\Theta v : T \quad \Gamma_1 \cdot k : \alpha\{y \leftarrow v\} \cdot \Gamma_2 \vdash_\Theta P : e \quad \Gamma_1 \cdot k : \uparrow [y : T] e' ; \alpha \cdot \Gamma_2 \vdash_\Theta \circ
\]
\[
\Gamma_1 \cdot k : \uparrow [y : T] e' ; \alpha \cdot \Gamma_2 \vdash_\Theta k! [v] ; P : e + e' \{y \leftarrow v\}
\]
Assume that \( \Gamma_1 = \Gamma_1 \cdot v : T \). Then note that \( v : T \) and \( k : \uparrow [y : T] e' ; \alpha \) are in condition of being exchanged since neither one depends on the other. However, when we attempt to exchange \( v : T \) and \( k : \alpha\{y \leftarrow v\} \) in the upper middle judgment we fail since \( \alpha\{y \leftarrow v\} \) may have free occurrences of \( v \).

Note that these issues do not appear in previous type-theoretic formulations of correspondence assertions for concurrent/distributed calculi since long term session types are not considered.

3 Safety Proof for \( \pi_s \)

In order to trace the execution of certain actions such as begin and end assertions we shall introduce a labeled transition semantics [10] (LTS) for \( \pi_s \). The LTS is defined modulo structural congruence \( \equiv \) and shall be used for formalizing the notion of safe process and showing that all typable processes with null effects are safe. The actions, denoted with letters \( \psi, \phi, \ldots \), of the transition system are:
\[
\begin{array}{c|c}
\text{fn}(\tau) & \text{def} \emptyset \\
\text{fn}(\text{begin } L) & \text{def} \text{ fn}(L) \\
\text{fn}(\text{end } L) & \text{def} \text{ fn}(L) \\
\text{fn}(\text{res}(a : T)) & \text{def} \{a\} \\
\text{fn}(\text{res}(k)) & \text{def} \{k\} \\
\text{gn}(\tau) & \text{def} \emptyset \\
\text{gn}(\text{begin } L) & \text{def} \emptyset \\
\text{gn}(\text{end } L) & \text{def} \emptyset \\
\text{gn}(\text{res}(a : T)) & \text{def} \{a\} \\
\text{gn}(\text{res}(k)) & \text{def} \{k\} \\
\end{array}
\]

\[
(\text{accept } a(k) \text{ in } P_1) \mid (\text{request } a(k) \text{ in } P_2) \xrightarrow{\tau} (\nu k)P_1 \mid P_2 \quad \text{Trans Link}
\]

\[
(k!v; P_1) \mid (k?(x) \text{ in } P_2) \xrightarrow{\tau} P_1 \mid P_2[x \leftarrow v] \quad \text{Trans Comm}
\]

\[
(k < n_i; P) \mid (k \triangleright \{l_1 : P_1 \sqcap \ldots \sqcap l_n : P_n\}) \xrightarrow{\tau} P \mid P_1, \text{ if } i \in 1..n \quad \text{Trans Brnc}
\]

\[
(\text{throw } k[k' ]; P_1) \mid (\text{catch } k(k') \text{ in } P_2) \xrightarrow{\tau} P_1 \mid P_2 \quad \text{Trans Catch}
\]

\[
\text{def } D \text{ in } (X[v]Q) \xrightarrow{\psi} \text{def } D \text{ in } (P[\bar{x} \leftarrow \bar{v}]Q), \quad \text{Trans Def1}
\]

\[
\begin{align*}
\text{begin } L; P \xrightarrow{\text{begin } L} P & \quad \text{Trans Begin} \\
\end{L; P \xrightarrow{\text{end } L} P & \quad \text{Trans End} \\
(\nu a : T)P \xrightarrow{\text{res}(a : T)} P & \quad \text{Trans ResN} \\
(\nu k)P \xrightarrow{\text{res}(k)} P & \quad \text{Trans ResCh} \\
\end{align*}
\]

\[
P \xrightarrow{\psi} P' \quad \text{Trans Def2}
\]

\[
\text{def } D \text{ in } P \xrightarrow{\psi} \text{def } D \text{ in } P' \\
P \xrightarrow{\psi} P' \quad \text{Trans Par, if } \text{gn}(\psi) \cap \text{fn}(Q) = \emptyset
\]

\[
\begin{align*}
P \mid Q \xrightarrow{\psi} P' \mid Q \quad \text{Trans \equiv} \\
P \equiv P' \quad P' \xrightarrow{\psi} Q' \quad Q' \equiv Q & \quad \text{Trans \equiv} \\
P \xrightarrow{\psi} Q & \\
\end{align*}
\]

Fig. 5. LTS for \(\pi_s\)

- \(P \xrightarrow{\text{begin } L} P'\) \(P\) reaches a \textit{begin } \(L\) assertion.
- \(P \xrightarrow{\text{end } L} P'\) \(P\) reaches a \textit{end } \(L\) assertion.
- \(P \xrightarrow{\text{res}(a : T)} P'\) \(P\) generates a new \textit{session name} \(a\).
- \(P \xrightarrow{\text{res}(k)} P'\) \(P\) generates a new \textit{channel name} \(k\).
- \(P \xrightarrow{\tau} P'\) \(P\) performs an internal action.

Thus the set of actions is \textit{begin } \(L\), \textit{end} \(L\), \textit{res}(a : T), \textit{res}(k), \tau\). The labeled transition system for \(\pi_s\) is given in Figure 5 we write \(P \xrightarrow{\psi} P'\) when \(P\) reduces
to $P'$ through action $\psi$. The same figure defines the free and generated names of an action.

A sequence of transitions may be tracked with traces. A trace $s$ is a sequence $\psi_1 \ldots \psi_n$ of actions. We use $e$ for the empty trace. The free names (resp. generated names) of a trace $\psi_1 \ldots \psi_n$ are defined as $\mathsf{fn}(\psi_1) \cup \ldots \cup \mathsf{fn}(\psi_1)$ (resp. $\mathsf{gn}(\psi_1) \cup \ldots \cup \mathsf{gn}(\psi_1)$). A traced transition is a sequence of actions:

**Definition 3.1** [Traced Transitions] $P$ reduces to $P'$ with trace $s$ if $P \xrightarrow{s} P'$, where $s$ is defined as:

$$P \equiv P' \Rightarrow P \xrightarrow{\epsilon} P' \quad \text{Trace } \equiv \quad P \xrightarrow{\psi} Q, Q \xrightarrow{s} P' \Rightarrow P \xrightarrow{\psi} P' \quad \text{Trace Action}$$

Note that we (by $\alpha$-conversion on process expressions) may assume that each generated name in a trace is distinct from all the others and from all the free names. In order to define when a process is safe we shall need to count the number of begin's and end's in traces. The former is defined as $\mathsf{begins}(\psi_1 \ldots \psi_n) \overset{\text{def}}{=} \mathsf{begins}(\psi_1) \cup \ldots \cup \mathsf{begins}(\psi_n)$ and the latter $\mathsf{ends}(\psi_1 \ldots \psi_n) \overset{\text{def}}{=} \mathsf{ends}(\psi_1) \cup \ldots \cup \mathsf{ends}(\psi_n)$, where $\cup$ stands for multi-set union and

$$\begin{align*}
\mathsf{begins}(\text{begin } L) & \overset{\text{def}}{=} (L) & \mathsf{ends}(\text{begin } L) & \overset{\text{def}}{=} (L) \\
\mathsf{begins}(\text{end } L) & \overset{\text{def}}{=} (\emptyset) & \mathsf{ends}(\text{end } L) & \overset{\text{def}}{=} (L) \\
\mathsf{begins}(\text{res}(u)) & \overset{\text{def}}{=} (\emptyset) & \mathsf{ends}(\text{res}(u)) & \overset{\text{def}}{=} (\emptyset) \\
\mathsf{begins}(\tau) & \overset{\text{def}}{=} (\emptyset) & \mathsf{ends}(\tau) & \overset{\text{def}}{=} (\emptyset)
\end{align*}$$

**Definition 3.2** [Safe Process] A process $P$ is safe if and only if for all traces $s$ and processes $P'$, if $P \xrightarrow{s} P'$ then $\mathsf{ends}(s) \leq \mathsf{begins}(s)$.

We now address the proof of safety, namely that a process typable with null effect is safe. This requires first showing that process reduction preserves typings and effects.

**Theorem 3.3** (Subject Congruence and Reduction) *Assume* $\Gamma \vdash_{\Theta} P : e$.

(i) *(subject congruence)* If $P \equiv Q$, then $\Gamma \vdash_{\Theta} Q : e$.

(ii) *(subject reduction)*

(a) If $P \xrightarrow{\tau} P'$, then $\Gamma' \vdash_{\Theta} P' : e$ where $\Gamma'$ and $\Gamma$ differ only in the effects assigned to the channel type $\bot$ (if any).

(b) If $P \xrightarrow{\text{begin } L} P'$, then $\Gamma \vdash_{\Theta} P' : e + (\langle L \rangle)$.

(c) If $P \xrightarrow{\text{end } L} P'$, then $\Gamma \vdash_{\Theta} P' : e - (\langle L \rangle)$ and $L \in e$.

(d) If $P \xrightarrow{\text{res}(a : T)} P'$ and $a \notin \mathsf{dom}(\Gamma)$, then $\Gamma \cdot a : T \vdash_{\Theta} P' : e$.

(e) If $P \xrightarrow{\text{res}(k)} P'$ and $a \notin \mathsf{dom}(\Gamma)$, then $\Gamma \cdot k : \bot_f \vdash_{\Theta} P' : e$ for some
effect $f$.

Subject Congruence is proved by induction on the derivation of $P \equiv Q$; the fact that effects are not lost when environments are composed (Def. 2.31) is crucial to its proof. Subject reduction is proved by cases according to action which takes place (see [3]). Finally, we may put the results together and obtain the main result. Its proof is based upon observing that the following invariant holds: If $\Gamma \vdash_P P : e$ and $P \xrightarrow{s} P'$ and $\Gamma(n) \cap \text{dom}(\Gamma) = \emptyset$, then $\text{ends}(s) \leq \text{begins}(s) + e$ (3).

Theorem 3.4 (Safety) If $\Gamma \vdash_P P : \langle \rangle$, then $P$ is a safe process.

By assigning the session names $a$ and $b$ the types indicated below, the good ATM (when executed concurrently with the Client and Bank) may be seen to be safe. However, with this type assignment Example 3.3 is not safe according to our type system as one might expect. Note that the necessary assertion labels are inserted as already explained in that example.

$$a : \sigma(\downarrow [idA : \text{Int}](\langle \rangle); &\{\text{deposit} : \downarrow [amtA : \text{Int}](\langle \langle idA, amtA \rangle \rangle); \uparrow [balA : \text{Int}](\langle \rangle); 1, \nabla \text{withdrawal} : \downarrow [amtA : \text{Int}](\langle \rangle); \uparrow [balA : \text{Int}](\langle \rangle); 1 \} (\langle \rangle))$$

$$b : \sigma(\&\{\text{deposit} : \downarrow [idB : \text{Int}](\langle \rangle); \downarrow [amtB : \text{Int}](\langle \langle idB, amtB \rangle \rangle); \uparrow [balB : \text{Int}](\langle \rangle); 1, \nabla \text{withdrawal} : \downarrow [idB : \text{Int}](\langle \rangle); \downarrow [amtB : \text{Int}](\langle \rangle); \uparrow [balB : \text{Int}](\langle \rangle); 1 \} (\langle \rangle))$$

where $\langle idA, amtA \rangle$ and $\langle idB, amtB \rangle$ are latent effects. See [3] for further details and more extensive examples including the use of effects in branch constructs.

4 Conclusions

This paper combines correspondence assertions and long term channel types. The latter are a versatile mechanism for restricting process behavior in multiplayer interactions. A session describes the message exchange pattern between two parties. However, these types provide no means of synchronization between sessions in a multi-session system. Indeed, we have shown an example illustrating how, when processing a clients request for a withdrawal operation, an ATM may decide either not to interact with the Bank at all, or withdraw less than the client requested for and at the same time deposit the rest in some other account (creating an unintended message exchange with the Bank). Session types are not expressive enough to distinguish these variants: In both these cases and also in the case of the “correct” ATM, the type assigned by the type system is exactly the same. By introducing correspondence assertions into the type system we are able to draw a fine line between them and identify the “correct” ATM from the faulty or malicious ones. A number of
issues require further attention:

• A benefit of our dependent type system is that constraint checks on the
code are possible. In particular, when a deposit operation is requested by
the client, correspondence assertions allow us to check that the account
number which the ATM communicates to the Bank is exactly the same
as the one punched in by the client as received by the ATM. It would
be interesting to enrich the set of assertion labels and thus consider more
expressive constraint equations.

• Session types look much like processes. In [15] a generic type system for the
\( \pi \)-calculus is studied in which types are CCS-like processes. They suggest
that it is possible to integrate a theory of correspondence assertions into
their framework. We are currently looking into this issue.

• Additional future work includes developing the formal theory of this calculus
in HOL [11] and using the development to encode and reason about security
and networking protocols.

Acknowledgments: We are grateful to the Laboratory for Secure Sys-
tems group at Stevens for interesting discussions, and in particular to Tom
Chothia for suggesting session types as a relevant concept. This work was
partially supported by The Stevens Technogenesis Fund, the NSF Grant No.
CCR-0220286 ITR:Secure Electronic Transactions, and the ARO Award No.
DAAD-19-01-1-0473.

References

[1] Antonio Vallecillo, V. V. and A. Ravara, Typing the behavior of objects and
component using session types, , 68 (2003).

and T. S. E. Maibaum, editors, Handbook of Logic in Computer Science:
Background - Computational Structures (Volume 2), Clarendon Press, Oxford,

[3] Bonelli, E., A. Compagnoni and E. Gunter, Correspondence
assertions for process synchronization in concurrent communications (2003),


message-passing programs, in: Proceedings of the 29th ACM SIGPLAN-SIGACT
Symposium on Principles of Programming Languages (2002).

[6] Clarke, E. and W. Marrero, Using formal methods for analyzing security,


A Auction Example

The following code models a small electronic auction system in a simple extension of our system supporting recursive channel types and channel types as parameters. The three main principals that conform the system are: 

- **Auctioneer**, **Seller** and **Buyer**. In a normal processing cycle a seller contacts the auctioneer informing of the product and initial bidding price desired. The auctioneer then waits to receive biddings from interested buyers. In our model, the auctioneer determines, after some fixed amount of time, that the bidding process is over and assigns the product to the highest bidder.

In order to keep the example simple we assume that an auctioneer can handle at most one seller at a time and that at least one bidder shall make a bid. The system shall begin operating once an initial seller and buyer are placed, this operation is taken care of by **InitAuctioneer**. After this, the main loop **Auctioneer** is called. The latter processes the three main operations **sell**, **bid** and **timeout**, as described below. Auxiliary processes that act as accumulators for the current seller and highest bidder are managed through a **SellerManager** and **BuyerManager**. Let us describe the full set of principals:

**InitAuctioneer.** The initialization process of the auctioneer begins by activating the seller manager and buyer manager processes, **SellerManager** and **BuyerManager**, after which it enters its main cycle implemented in the process **Auctioneer**.

\[
\text{InitAuctioneer}(sAuc, sBM, sSM) = \text{request } sBM(kBM) \text{ in } \\
\text{request } sSM(kSM) \text{ in } \\
\text{Auctioneer}[sAuc, kBM, kSM]
\]

\(sAuc\) is a session name for interaction with **Auctioneer**, \(sBM\) is a session name for interaction with **BuyerManager**, and \(sSM\) a session name for interaction with **SellerManager**.

**Auctioneer.** The auctioneer waits to receive requests for one of three operations:

- **sell**: This is invoked by a seller. Since the auctioneer can handle at most one seller, it lets the seller manager know that it must cancel the previous seller (in turn the seller manager shall contact this seller to let her/him know). After that, it reads the product and base price that the seller wishes to commence with, it informs the **BuyerManager** that a new base price is in effect and finally passes on the new seller (i.e. \(k\)) to the seller manager **SellerManager**.

- **bid**: This is invoked by a buyer. The auctioneer informs the buyer manager **BuyerManager** that a new bidder has arrived and passes on the bidder (i.e. \(k\)) to this manager.

- **timeout**: This operation is invoked when no further bidding time is left and hence that the current highest bidder has successfully acquired the
item sold. It informs the seller manager \texttt{SellerManager} and the buyer manager \texttt{BuyerManager} of this situation.

\texttt{Auctioneer}(sAuc, kBm, kSM) =
\begin{algorithmic}
  \STATE accept \texttt{sAuc}(k) in
  \STATE $k \triangleright \{ \text{sell: } kSM \triangleright \text{cancel}; \text{ prod, basePrice in kBm} \triangleright \text{newProduct};$
  \STATE \hspace{1cm} kBm!(\text{prod, basePrice}); \text{ throw kSM}[k]; \text{ Auctioneer}[sAuc, kBm, kSM],$
  \STATE \hspace{1cm} \Box \text{bid: } kBm \triangleright \text{newBidder}; \text{ throw kBm}(k); \text{ Auctioneer}[sAuc, kBm, kSM],$
  \STATE \hspace{1cm} \Box \text{timeout: } kSM \triangleright \text{sold}; kBm \triangleright \text{bought}; \text{ Auctioneer}[sAuc, kBm, kSM] \}$
\end{algorithmic}

\textbf{SellerManager}. The seller manager acts as an accumulator which holds the channel to the current seller that the auctioneer is dealing with. The auctioneer instructs it to do two possible things:
- forward to the seller the message that her item has been sold or
- forward to the seller the auction has been canceled due to the arrival of a new seller and read in the new seller.

\texttt{SellerManager}(kSM, h) =
\begin{algorithmic}
  \STATE $kSM \triangleright \{ \text{sold: } h \triangleright \text{sold}; \text{ SellerManager}[kSM, dummySeller],$
  \STATE \hspace{1cm} \Box \text{cancel: } h \triangleright \text{cancel}; \text{ catch kSM}(h') \text{ in SellerManager}[kSM, h'] \}$
\end{algorithmic}

\textbf{BuyerManager}. The buyer manager acts as an accumulator which holds the channel to the current seller that the auctioneer is dealing with.

\texttt{BuyerManager}(kBm, h, prod, currBid) =
\begin{algorithmic}
  \STATE $kBm \triangleright \{ \text{newProduct: } kBm!(\text{prod, basePrice});$
  \STATE \hspace{1cm} \text{ BuyerManager}[kBm, dummyBuyer, prod, basePrice],$
  \STATE \hspace{1cm} \Box \text{newBidder: } \text{ catch kBm}(hAux) \text{ in } hAux!(\text{prod, bid}) \text{ in}$
  \STATE \hspace{2cm} if bid > currBid
  \STATE \hspace{3cm} then h < outBidded;
  \STATE \hspace{3cm} \text{ BuyerManager}[kBm, hAux, prod, bid]$
  \STATE \hspace{1cm} else hAux \triangleright \text{tooLow};$
  \STATE \hspace{1cm} \text{ BuyerManager}[kBm, h, prod, currBid]$
  \STATE \hspace{1cm} \Box \text{bought: } h \triangleright \text{bought}; \text{ BuyerManager}[kBm, dummyBuyer, *, 0] \}$
\end{algorithmic}

It waits to receive one of the following selections:
- \texttt{newProduct}: this is selected by the auctioneer and informs that a new seller has arrived and passes on the product and base price of this product.
- \texttt{newBidder}: this is selected by the auctioneer when a new bidder has arrived. It delegates the bidder to \texttt{BuyerManager}. The latter reads in the bid and compares it to its current highest bid: if the former is greater than the latter then it informs the current highest bidder (i.e. $h$) that it has been outbid and recursively calls itself with the new bidder as a parameter; otherwise the new bidder is informed that her bid is too low and \texttt{BuyerManager} recursively calls itself with the current highest bidder
as the highest bidder for the call.

- **bought.** this is selected by the auctioneer to inform the buyer manager that the current highest bidder has successfully acquired the product.

**Seller.** This process describes the behavior of a seller. She requests a session with the auctioneer and lets her know that she is willing to sell a product `prod` at price `price`. She then waits to be informed whether her product was sold or the auction was canceled due to the arrival of some new seller:

\[
\text{Seller}(sAuc, prod, price) = \text{request } sAuc(k) \text{ in } k < \text{sell; } k![\text{prod, price}];
\]

\[
k \triangleright \{\text{sold: inact, cancel: inact} \}
\]

**Buyer.** The buyer requests a session with the auctioneer and selects a bidding operation. She then throws the product she is interested in and the price she is willing to pay and awaits one of three possible replies:

- **outBidded:** In some later cycle a new bidder has outbid her.
- **bought:** She has successfully bought the product.
- **tooLow:** Her initial bid was too low and thus rejected.

\[
\text{Buyer}(sAuc, prod, price) = \text{request } sAuc(k) \text{ in } k < \text{bid; } k![\text{prod, price}];
\]

\[
k \triangleright \{\text{outBidded: inact, bought: inact, tooLow: inact} \}
\]

Note that in this example we do not take into account error capture and recovery such as when a bidder attempts to make a bid for an item which has not been placed for selling.

Let `P` be the process `def D in Q` where `Q` is:

\[
\text{InitAuctioneer}[a, sBM, sSM] | \text{InitBuyerManager}[sBM, dummyBuyer] | \text{InitSellerManager}[sSM, dummySeller] | \text{Buyer}[sAuc, prod, bid] | \text{Seller}[sAuc, prod, price]
\]

and where process declarations `D` are those given above. Assuming that the theory of correspondence assertions is not exploited we may type `P` under `Γ` (i.e. `Γ ⊢_0 P : []`) is derivable) where the environment `Γ` is defined as follows:

\[
Γ = sAuc : σ(α), sBM : σ(β), sSM : σ(γ), d DummyBuyer : @\{\text{outBidded: 1, bought: 1, tooLow: 1}\}, d DummySeller : @\{\text{sold: 1, cancel: 1}\}
\]

The channel types `α`, `β` and `γ` are described in Figure A.2.

Note that all effects are empty (hence have been omitted for the sake of readability) and thus `P` is (trivially) safe. The channels `dummyBuyer` and `dummySeller` represent dummy connections. Their unique objective is to initialize the “accumulators” `BuyerManager` and `SellerManager`, as already
\begin{verbatim}
InitAuctioneer(sAuc, sBM, sSM) =
    request sBM(kBM) in request sSM(kSM) in Auctioneer[sAuc, kBM, kSM]

InitBuyerManager(sBM, dummyBuyer) =
    accept sBM(kBM) in BuyerManager[kBM, dummyBuyer, *, 0]

InitSellerManager(sSM, dummySeller) =
    accept sSM(kSM) in SellerManager[kSM, dummySeller]

Auctioneer(sAuc, kBM, kSM) =
    accept sAuc(k) in
    k ▷ {sell: kSM ▷ cancel; k?(prod, basePrice) in kBM ▷ newProduct;
        kBM!(prod, basePrice); throw kSM[k]; Auctioneer[sAuc, kBM, kSM],
        □ bid: kBM ▷ newBidder; throw kBM(k); Auctioneer[sAuc, kBM, kSM],
        □ timeout: kSM ▷ sold; kBM ▷ bought; Auctioneer[sAuc, kBM, kSM] }

BuyerManager(kBM, h, prod, currBid) =
    kBM ▷ {newProduct: kBM?(prod, basePrice);
        BuyerManager[kBM, dummyBuyer, prod, basePrice],
        □ newBidder: catch kBM(hAux) in hAux?(prod, bid) in
        if bid > currBid
            then h ▷ outBidden;
            BuyerManager[kBM, hAux, prod, bid]
        else hAux ▷ tooLow;
            BuyerManager[kBM, h, prod, currBid]
        □ bought: h ▷ bought; BuyerManager[kBM, dummyBuyer, *, 0] }

SellerManager(kSM, h) =
    kSM ▷ {sold: h ▷ sold; SellerManager[kSM, dummySeller],
        □ cancel: h ▷ cancel; catch kSM(h') in SellerManager[kSM, h'] }

Seller(sAuc, prod, price) = request sAuc(k) in k ▷ sell; k!(prod, price);
    k ▷ {sold: inact,
        □ cancel: inact }

Buyer(sAuc, prod, price) = request sAuc(k) in k ▷ bid; k!(prod, price);
    k ▷ {outBidden: inact,
        bought: inact,
        tooLow: inact }
\end{verbatim}

Fig. A.1. Full code for the auction example.

We shall now consider the following property of interest to the buyer: to verify that if she puts forward a bid then this bid is taken into account by
\[
\alpha = \&\{sell :>[\text{Int,Int}] ; \oplus \{sold : 1, cancel : 1\} ; 1, \\
\text{bid} : \downarrow [\text{Int,Int}] ; \oplus \{outBidded : 1, bought : 1, tooLow : 1\}, \\
\text{timeout} : 1\}
\]
\[
\beta = \mu.t.\&\{\text{newProduct} : \downarrow [\text{Int,Int}] ; t, \\
\text{newBidder} : \downarrow [\downarrow [\text{Int,Int}] ; \oplus \{outBidded : 1, bought : 1, tooLow : 1\}] ; t, \\
bought : t\}
\]
\[
\gamma = \mu.t.\&\{\text{sold} : t, cancel : \downarrow [\oplus \{\text{sold} : 1, cancel : 1\}] ; t\}
\]

Fig. A.2. Channel types for \(\Gamma\)

\[
\text{Buyer(\text{sauc}, prod, price)} = \text{request sauc}(k) \in k\triangleleft \text{bid}; k! [\text{prod, price}]; \\
k \triangleright \{\text{outBidded} : \text{end} ([\bullet]); \text{inact}, \\
\text{bought} : \text{end} ([\bullet]); \text{inact}, \\
\text{tooLow} : \text{end} ([\bullet]); \text{inact}\}
\]

\[
\text{BuyerManager(kBM, h, prod, currBid)} = \\
kBM \triangleright \{\text{newProduct} : \text{kBM?(prod, basePrice)}; \\
\text{BuyerManager[kBM, dummyBuyer, prod, basePrice]}, \\
\boxed{\text{newBidder} : \text{catch kBM(hAux) in hAux?(prod, bid) in begin} ([\bullet]); \\
\text{if bid} > \text{currBid} \\
\text{then h} \triangleleft \text{outBidded}; \\
\text{BuyerManager[kBM, hAux, prod, bid]} \\
\text{else hAux} \triangleleft \text{tooLow}; \\
\text{BuyerManager[kBM, h, prod, currBid]} \\
\boxed{\text{bought} : \text{begin} ([\bullet]); h \triangleleft \text{bought}; \\
\text{BuyerManager[kBM, dummyBuyer, *, 0] }}
\]

Fig. A.3. Code augmented with assertions.

The Auctioneer, in other words it is not ignored. The latter may be verified by checking that the buyer accumulator holds the connection to the buyer at some moment. This may be accomplished by our theory of correspondence assertions as follows. First we insert an end assertion into each of the branches of the buyer’s code. The resulting code is depicted at the top of Figure A.3. Then begin assertions in the accumulator for buyers is introduced as depicted in the same figure. Note that no assertions are placed in the newProduct branch since no interaction with the buyer takes place here.

Note also that in the augmented buyer’s code since the only “input” received from k is the branching operation this example shows how effects in branch types may be used.

The types assigned to the names in the environment \(\Gamma\) must now be aug-
mented with effects as follows:

\[ \Gamma' = s\text{Auc} : \sigma(\alpha'), s\text{BM} : \sigma(\beta'), s\text{SM} : \sigma(\gamma), \]
\[ \text{dummyBuyer} : \oplus\{\text{outBidded} : 1, \text{bought} : 1, \text{tooLow} : 1\}\{\bullet\}, \]
\[ \text{dummySeller} : \oplus\{\text{sold} : 1, \text{cancel} : 1\}\{\emptyset\} \]
\[ \alpha' = \&\{\text{sell} \downarrow [\text{Int}, \text{Int}] ; \oplus\{\text{sold} : 1, \text{cancel} : 1\}; 1, \]
\[ \text{bid} : \uparrow [\downarrow [\text{Int}, \text{Int}]\emptyset] ; \oplus\{\text{outBidded} : 1, \text{bought} : 1, \text{tooLow} : 1\}\{\bullet\} ; \emptyset, \]
\[ \text{timeout} : 1\emptyset\emptyset \]
\[ \beta' = \mu t. &\{\text{newProduct} \downarrow [\text{Int}, \text{Int}]\emptyset\} ; t, \]
\[ \text{newBidder} : \downarrow [\downarrow [\text{Int}, \text{Int}]\emptyset] ; \oplus\{\text{outBidded} : 1, \text{bought} : 1, \text{tooLow} : 1\}\{\bullet\} ; \emptyset ; t, \]
\[ \text{bought} : t\emptyset\emptyset \]

The fact that the resulting system \( P' \) is safe indicates that at some point the \text{BuyerManager} must have issued one of the operations \text{outBidded}, \text{bought} or \text{tooLow}. As a consequence, it cannot have ignored the buyers bid.

## B  Safety: Proofs

**Definition B.1** [Free names of types] The set of free names of a type \( U \), written \( \text{fn}(U) \), is defined as follows:

\[
\begin{align*}
\text{fn}(\text{Int}) & \overset{\text{def}}{=} \emptyset \\
\text{fn}(\sigma(\alpha)) & \overset{\text{def}}{=} \text{fn}(\alpha) \\
\text{fn}(\uparrow [x : U]e ; \alpha) & \overset{\text{def}}{=} \text{fn}(U) \cup (\text{fn}(e) \cup \text{fn}(\alpha)) \setminus \{x\} \\
\text{fn}(\&\{l_1 : \alpha_1, \ldots, l_n : \alpha_n\}e) & \overset{\text{def}}{=} (\bigcup_{i=1..n} \text{fn}(\alpha_i)) \cup \text{fn}(e) \\
\text{fn}(\oplus\{l_1 : \alpha_1, \ldots, l_n : \alpha_n\}e) & \overset{\text{def}}{=} (\bigcup_{i=1..n} \text{fn}(\alpha_i)) \cup \text{fn}(e) \\
\text{fn}(1) & \overset{\text{def}}{=} \emptyset \\
\text{fn}(\perp_e) & \overset{\text{def}}{=} \text{fn}(e) \\
\text{fn}(\langle L_1, \ldots, L_n \rangle) & \overset{\text{def}}{=} \bigcup_{i=1..n} \text{fn}(L_i) \\
\text{fn}(\langle v_1, \ldots, v_n \rangle) & \overset{\text{def}}{=} \{v_j \mid v_j \text{ name }\}
\end{align*}
\]

Let \( \mathcal{F} \) stand for \( P : e, x : U, n : \text{Int} \) or \( (\bar{v}) : (\bar{x} : \bar{U}) \).

**Lemma B.2** (i) \( \circ \) is partially commutative

(ii) \( \circ \) is partially associative: if \( \Gamma_1 \times \Gamma_2 \) and \( \Gamma_2 \times \Gamma_3 \) and \( \Gamma_1 \times \Gamma_2 \circ \Gamma_3 \), then

24
(Γ₁ ∘ Γ₂) ∘ Γ₃ × Γ₁ ∘ (Γ₂ ∘ Γ₃).

(iii) Γ₁ ∘ Γ₂ × Γ₃ and Γ₁ × Γ₂ implies Γ₂ × Γ₃ and Γ₁ ∘ Γ₂ ∘ Γ₃.

(iv) Γ₁ × Γ₂ ∘ Γ₃ and Γ₂ × Γ₃ implies Γ₁ × Γ₂ and Γ₁ ∘ Γ₂ ∘ Γ₃.

Proof.

(i) If Γ ∼ Γ', then Γ' ∼ Γ and a close inspection of Def. [2,3] yields the desired result.

(ii) As for associativity, we proceed by induction on the length of Γ₁.

(iii) By induction on the length of Γ₁.

(iv) By induction on the length of Γ₂.

Lemma B.3 (Subsumption Elimination) If Γ ⊢ θ P : e, then for some e' ≤ e, Γ ⊢ θ P : e' is derivable without using the rule Type Subsum. Moreover, e' is the minimum effect for P in Γ in the following sense: for all e'', if Γ ⊢ θ P : e'', then e' ≤ e''.

Proof. By induction on the derivation of Γ ⊢ θ P : e using the properties of ≤ on multisets.

Lemma B.4 If Γ · k : 1 ⊢ θ P : e, then Γ · k : ⊥ₗ ⊢ θ P : e for any effect e such that fn(e) ⊆ dom(Γ).

Proof. By induction on the length of the derivation of Γ · k : 1 ⊢ θ P : e. Some cases which are worth commenting on are:

Type Snd. When applying the induction hypothesis to the upper judgement (Γ ⊢ θ v : T) of the Type Snd rule note that T is a plain type (and not a channel type).

Type Stop. Note that the condition ranCh(Γ) ⊆ {1, ⊥ₗ} is preserved.

Type Thr. The condition β ≠ 1 in the formulation of Type Thr is required for this case to go through.

Type Par. If k : 1 ∈ Γ₁ ∘ Γ₂, then either k : 1 ∈ Γ₁ \ Γ₂ or k : 1 ∈ Γ₂ \ Γ₁. We thus apply the induction hypothesis to the appropriate upper judgement. Note that the resulting judgement shall still be compatible with the other upper judgement of Type Par.

Lemma B.5 (Derivability implies well-formedness of environments) If Γ ⊢ θ J, then Γ ⊢ θ ϕ.

Proof. By induction on the length of the derivation of Γ ⊢ θ J. We consider some cases in order to illustrate the arguments used:

Type Acpt. We use the induction hypothesis and condition C1.
**Type Par.** We use the fact that $\Gamma_1 \vdash_\Theta \circ$, $\Gamma_1 \vdash_\Theta \circ$ and $\Gamma_1 \triangleleft \Gamma_2$, implies $\Gamma_1 \circ \Gamma_2 \vdash_\Theta \circ$. 

The following Weakening Lemma holds without restrictions for plain types. However, in the case of channel types we must require the new type introduced into the environment to be $1$ or $\bot_e$. This is necessary due to the Type Stop and Type PVar typing rules.

**Lemma B.6 (Weakening)** If $\Gamma \vdash_\Theta J$ and $x \notin \text{dom}(\Gamma)$ and $\Gamma \cdot x : U \vdash_\Theta \circ$, then

(i) $\Gamma \cdot x : U \vdash_\Theta J$, if $U$ is a plain type.

(ii) $\Gamma \cdot x : U \vdash_\Theta J$, if $U$ is a channel type and $U = 1$ or $U = \bot_e$.

**Proof.** By induction on the length of the derivation of $\Gamma \vdash_\Theta J$. In the Type Par rule, if $U$ is a plain type we apply the induction hypothesis on both of its upper judgements; if $U$ is a channel type we select (any) one of the two and apply the induction hypothesis. 

**Lemma B.7 (Exchange)** If $\Gamma \cdot x : U \cdot x' : U' \cdot \Gamma' \vdash_\Theta J$, then $\Gamma \cdot x' : U' \cdot x : U \cdot \Gamma' \vdash_\Theta J$.

**Proof.** This is immediate from the fact that our environments are sets of assumptions.

**Lemma B.8 (Redundant Hypothesis Elimination)** If $\Gamma \cdot x : U \vdash_\Theta J$ and $x \notin \text{fn}(\Gamma, J)$, then $\Gamma \vdash_\Theta J$.

**Proof.** By induction on the length of the derivation of $\Gamma \cdot x : U \vdash_\Theta J$. 

**Proof.** [Proof of Subject Congruence] By induction on the derivation of $P \equiv Q$ we prove that:

(i) $\Gamma \vdash_\Theta P : e$ implies $\Gamma \vdash_\Theta Q : e$

(ii) $\Gamma \vdash_\Theta Q : e$ implies $\Gamma \vdash_\Theta P : e$

Structural congruence is the smallest equivalence relation satisfying the rules of Figure B.1. By Subsumption Elimination (Lemma B.3) we shall assume that $\Gamma \vdash_\Theta P : e$ is derivable without using Type Subsum, in the first item, and similarly for $\Gamma \vdash_\Theta Q : e$ in the second item.

**SC Refl, SC Symm, SC Trans.** Trivial.

**SC Stop.** (i) Suppose $\Gamma \vdash_\Theta P : e$. Let $\Gamma'$ be the subset of $\Gamma$ consisting of its plain assumptions. Then we construct the derivation

\[
\begin{array}{c}
\Gamma \vdash_\Theta P : e \\
\hline
\Gamma' \vdash_\Theta \circ \\
\hline
\Gamma' \vdash_\Theta \text{stop} : \emptyset \\
\hline
\Gamma' \circ \Gamma' \vdash_\Theta \text{stop} : e
\end{array}
\]

Type Stop

Type Par
\[ P \equiv P \quad \text{SC Refl} \]
\[ P \equiv Q \Rightarrow Q \equiv P \quad \text{SC Symm} \]
\[ P \equiv Q, Q \equiv R \Rightarrow P \equiv R \quad \text{SC Trans} \]

\[ P|_{\text{stop}} \equiv P \quad \text{SC Stop} \]
\[ P|_{Q} \equiv Q|_{P} \quad \text{SC Par Comm} \]
\[ (P|_{Q})|_{R} \equiv P|_{(Q|R)} \quad \text{SC Par Asoc} \]

\[ P \equiv P' \Rightarrow (\nu u)P \equiv (\nu u)P' \quad \text{SC New Name/Chan} \]
\[ P \equiv P' \Rightarrow P|_{Q} \equiv P'|_{Q} \quad \text{SC Par} \]
\[ P \equiv P' \Rightarrow k?(y) \text{in } P \equiv k?(y) \text{in } P' \quad \text{SC Rcv} \]
\[ P \equiv P' \Rightarrow k![v]; P \equiv k![v]; P' \quad \text{SC Send} \]
\[ P \equiv P' \Rightarrow \text{accept } a(k) \text{ in } P \equiv \text{accept } a(k) \text{ in } P' \quad \text{SC Acpt} \]
\[ P \equiv P' \Rightarrow \text{request } a(k) \text{ in } P \equiv \text{request } a(k) \text{ in } P' \quad \text{SC Requ} \]
\[ P \equiv P' \Rightarrow k \triangleleft l; P \equiv k \triangleleft l; P' \quad \text{SC Sel} \]
\[ P \equiv P' \Rightarrow k \triangleright \{\ldots \triangleleft l_{i} : P \triangleleft \ldots \} \equiv k \triangleright \{\ldots \triangleleft l_{i} : P' \triangleleft \ldots \} \quad \text{SC Brnch} \]
\[ P \equiv P' \Rightarrow \text{throw } k[k']; P \equiv \text{throw } k[k']; P' \quad \text{SC Thr} \]
\[ P \equiv P' \Rightarrow \text{catch } k(k') \text{ in } P \equiv \text{catch } k(k') \text{ in } P' \quad \text{SC Cat} \]
\[ P \equiv P' \Rightarrow \text{def } D \text{ in } P \equiv \text{def } D \text{ in } P' \quad \text{SC Def} \]
\[ P \equiv P' \Rightarrow \text{begin } L; P \equiv \text{begin } L; P' \quad \text{SC Begin} \]
\[ P \equiv P' \Rightarrow \text{end } L; P \equiv \text{end } L; P' \quad \text{SC End} \]

\[(\nu u_{1} : T_{1})(\nu u_{2} : T_{2})P \equiv (\nu u_{2} : T_{2})(\nu u_{1} : T_{1})P, \quad \text{SC Res Res} \]
\[\text{if } u_{1} \neq u_{2}, u_{1} \notin \text{fn}(T_{2}), u_{2} \notin \text{fn}(T_{1}) \]
\[(\nu u)(P|Q) \equiv (\nu u)P|_{Q}, \text{ if } u \notin \text{fn}(Q) \quad \text{SC Res Par} \]
\[(\nu u)\text{def } D \text{ in } P \equiv \text{def } D \text{ in } (\nu u)P, \text{ if } u \notin \text{fn}(D) \quad \text{SC Res Def} \]

\[(\text{def } D \text{ in } P)|_{Q} \equiv \text{def } D \text{ in } (P|Q), \text{ if } \text{fpv}(D) \cap \text{fpv}(Q) = \emptyset \quad \text{SC Def Par} \]
\[\text{def } D \text{ in } \text{def } D' \text{ in } P \equiv \text{def } D \text{ and } D' \text{ in } P, \text{ if } \text{fpv}(D) \cap \text{fpv}(D') = \emptyset \quad \text{SC Def And} \]

---

Fig. B.1. Structural Congruence (full definition)

27
Note that
(a) $\Gamma' \vdash_{\Theta} P : e$ follows from $\Gamma \vdash_{\Theta} P : e$ and Lemma B.3.
(b) $\Gamma \simeq \Gamma'$ and $\Gamma \circ \Gamma' = \Gamma$.
(ii) Suppose $\Gamma \simeq \Gamma'$ in

$$
\frac{
\Gamma' \vdash_{\Theta} \circ \\
\Gamma \vdash_{\Theta} P : e
}{
\Gamma \circ \Gamma' \vdash_{\Theta} P \text{ stop : } e
} \quad \text{Type Stop}
$$

$$
\frac{
\Gamma \vdash_{\Theta} \circ \\
\Gamma \circ \Gamma' \vdash_{\Theta} P \text{ stop : } e
}{
\Gamma \vdash_{\Theta} P \text{ stop : } e
} \quad \text{Type Par}
$$

We want to show that $\Gamma \circ \Gamma' \vdash_{\Theta} P : e$ from $\Gamma \vdash_{\Theta} P : e$. Note that $\Gamma \circ \Gamma'$ differs from $\Gamma$ only in the types of the channel names. For any $k : \alpha \in \Gamma \circ \Gamma'$ either
- $\alpha = \bot_e$ and $k : \beta \in \Gamma$ and $k : \overline{\beta} \in \Gamma'$. Note that by the restriction on the Type Stop rule, $\beta = 1$ is the only possible case and thus $e = (\emptyset)$.
- We apply Lemma B.3 and conclude.
- $\alpha = \bot_e$ and $k : \bot_e \in \Gamma$ and $k \notin \text{dom}(\Gamma)$. In this case we are done.
- $\alpha = \bot_e$ and $k \notin \text{dom}(\Gamma)$ and $k : \bot_e \in \Gamma'$. We apply Weakening (Lemma B.6) and conclude.
- $\alpha = 1$ and $k : 1 \in \Gamma$ and $k \notin \text{dom}(\Gamma')$. In this case we are done.
- $\alpha = 1$ and $k \notin \text{dom}(\Gamma)$ and $k : 1 \in \Gamma'$. We conclude using Weakening (Lemma B.6).

**SC Par Comm.** This relies on the symmetry of $\simeq$ and commutativity of $\circ$.

**SC Par Asoc.** This relies on associativity of $\circ$, Lemma B.2 (3) and Lemma B.2 (4).

**SC New Name/Chan - SC End.** These cases rely on the induction hypothesis.

**SC Res Res.** Suppose

$$
\frac{
\Gamma \cdot u_1 : T_1 \cdot u_2 : T_2 \vdash_{\Theta} P : e \\
\Gamma \cdot u_1 : T_1 \vdash_{\Theta} \circ
}{
\Gamma \cdot u_1 : T_1 \vdash_{\Theta} (\nu u_2 : T_2)P : e
} \quad \Gamma \vdash_{\Theta} \circ
$$

Now note that:
- From $\Gamma \cdot u_1 : T_1 \cdot u_2 : T_2 \vdash_{\Theta} P : e$ and the Exchange Lemma (Lemma B.7) we deduce $\Gamma \cdot u_2 : T_2 \cdot u_1 : T_1 \vdash_{\Theta} P : e$.
- From Lemma B.3 and $\Gamma \cdot u_1 : T_1 \cdot u_2 : T_2 \vdash_{\Theta} P : e$ we deduce $\Gamma \cdot u_1 : T_1 \cdot u_2 : T_2 \vdash_{\Theta} \circ$. Using this fact, together with $\Gamma \cdot u_1 : T_1 \vdash_{\Theta} \circ$ and $\Gamma \vdash_{\Theta} \circ$ and $u_1 \notin \text{fn}(T_2)$ and $u_2 \notin \text{fn}(T_1)$, we may obtain $\Gamma \cdot u_2 : T_2 \vdash_{\Theta} \circ$. Thus we construct the derivation

$$
\frac{
\Gamma \cdot u_2 : T_2 \cdot u_1 : T_1 \vdash_{\Theta} P : e \\
\Gamma \cdot u_2 : T_2 \vdash_{\Theta} \circ
}{
\Gamma \cdot u_2 : T_2 \vdash_{\Theta} (\nu u_1 : T_1)P : e
} \quad \Gamma \vdash_{\Theta} \circ
$$

The second item is similar.

**SC Res Par.** Since channel restriction $(\nu k)$ is dealt with in a similar way to
type restriction \((\nu a : T)\), we only develop the latter.

(i) Suppose

\[
\Gamma_1 \cdot a : T \vdash_\Theta P : e_P \quad \Gamma_2 \cdot a : T \vdash_\Theta Q : e_Q \quad (\Gamma_1 \circ \Gamma_2) \cdot a : T \vdash_\Theta \circ(\cdot)
\]

\[
\begin{array}{c}
\frac{\Gamma_1 \circ \Gamma_2 \vdash_\Theta \circ(\cdot) (\nu a : T)(P \mid Q) : e_P + e_Q}{\frac{\Gamma_1 \circ \Gamma_2 \vdash_\Theta \circ(\cdot)(\nu a : T)P \mid Q : e_P + e_Q}{\frac{\Gamma_1 \cdot a : T \vdash_\Theta P : e_P \quad \Gamma_2 \cdot a : T \vdash_\Theta Q : e_Q}{\Gamma_1 \circ \Gamma_2 \vdash_\Theta \circ(\cdot) (\nu a : T)P \mid Q : e_P + e_Q}}}
\end{array}
\]

where \(\Gamma_1 \cdot a : T \times \Gamma_2 \cdot a : T\) and \(a \notin \text{fn}(Q)\). Note that from \(\Gamma_1 \circ \Gamma_2 \vdash_\Theta \circ(\cdot)\) we may deduce that \(\Gamma_1 \vdash_\Theta \circ(\cdot)\) by contradiction as follows: Suppose that \(\Gamma_1\) is not a well-formed environment; this may be due to two reasons:

(a) If there are cyclic dependencies, then since no linear dependencies are allowed the cycle must be formed by assumptions of the form \(a_i : T_i\), in which case they also constitute a cycle in \(\Gamma_1 \circ \Gamma_2\).

(b) Suppose there exists \(x : U \in \Gamma_1\) such that \(\text{fn}(U) \nsubseteq \text{dom}(\Gamma_1) \setminus \{x\}\). Note that it is not possible for \(U\) to be a plain type since we arrive at a contradiction immediately from the fact that \(\Gamma_1 \circ \Gamma_2 \vdash_\Theta \circ(\cdot)\). Therefore, \(U\) must be a channel type \(\alpha\).

• If \(x : \alpha \notin \text{dom}(\Gamma_2)\), then \(x : \alpha \in \Gamma_1 \circ \Gamma_2\). Since \(\text{domP}_1(\Gamma_1) = \text{domP}_1(\Gamma_1 \circ \Gamma_2)\) we conclude that \(\Gamma_1 \circ \Gamma_2\) is not well-formed and thus reach a contradiction.

• If \(x : \overline{\alpha} \in \text{dom}(\Gamma_2)\), then we reason as follows. Let \(y \in \text{fn}(\alpha)\) such that \(y \notin \text{dom}(\Gamma_1) \setminus \{x\}\). Note that \(y \in \text{fn}(\perp_e)\), where \(\perp_e\) is the result of composing \(\alpha\) and \(\overline{\alpha}\). Since \(\text{domP}_1(\Gamma_1) = \text{domP}_1(\Gamma_1 \circ \Gamma_2)\) we conclude that \(\Gamma_1 \circ \Gamma_2\) is not well-formed and thus reach a contradiction. Therefore, it must be the case that \(\Gamma_1 \vdash_\Theta \circ(\cdot)\).

Then we may construct the derivation

\[
\begin{array}{c}
\frac{\Gamma_1 \cdot a : T \vdash_\Theta P : e_P \quad \Gamma_1 \vdash_\Theta \circ(\cdot)(\nu a : T)P : e_P}{\Gamma_1 \circ \Gamma_2 \vdash_\Theta \circ(\cdot)(\nu a : T)P \mid Q : e_P + e_Q}
\end{array}
\]

(ii) For the second item we may assume that the derivation ends as indicated below:

\[
\begin{array}{c}
\frac{\Gamma_1 \cdot a : T \vdash_\Theta P : e_P \quad \Gamma_1 \vdash_\Theta \circ(\cdot)(\nu a : T)P : e_P}{\Gamma_1 \circ \Gamma_2 \vdash_\Theta \circ(\cdot)(\nu a : T)P \mid Q : e_P + e_Q}
\end{array}
\]

From Lemma 3.3 we deduce that \(\Gamma_1 \cdot a : T \vdash_\Theta \circ(\cdot)\) and \(\Gamma_2 \vdash_\Theta \circ(\cdot)\). Moreover, since \(\Gamma_1 \times \Gamma_2\) we know that \(\Gamma_2 \cdot a : T \vdash_\Theta \circ(\cdot)\). We may thus apply Weakening (Lemma 3.4) to obtain \(\Gamma_2 \cdot a : T \vdash_\Theta \circ(\cdot)Q : e_Q\). Similarly, we have \((\Gamma_1 \circ \Gamma_2) \cdot a : T \vdash_\Theta \circ(\cdot)\). This gives us all the tools to construct a derivation of the intended judgement.

**SC Res Def.** We develop the case \(u\) is a session name \(a\), the case where \(u\) is a channel name \(k\) is developed similarly.

(i) For the first item the derivation of \(\Gamma \vdash_\Theta \langle \nu a : T \rangle \text{def} D \in Q : e\) must
be of the form:

\[ \Gamma \cdot a : T \vdash_{\Theta \setminus x} D \text{ in } Q : e \quad \Gamma \vdash_{\Theta \setminus x} \phi \]  

\[ \Gamma \vdash_{\Theta \setminus x} (\nu a : T)\text{def } D \text{ in } Q : e \]  

where the top left-hand judgement results from

\[ (\Gamma \cdot a : T) \setminus \text{chan}(\Gamma) \cdot \bar{x}_i : \bar{T}_i \vdash_{\Theta} P_i : \emptyset \quad \text{fn}(\bar{T}_i) \cap \text{dom}(\Gamma) = \emptyset \quad \Theta(X_i) = (\bar{x}_i : \bar{T}_i) \quad \Gamma \cdot a : T \vdash_{\Theta} Q \]  

\[ \Gamma \cdot a : T \vdash_{\Theta \setminus x} \text{def } D \text{ in } Q : e \]  

We apply the Redundant Hypothesis Lemma (Lemma 2.8) to \((\Gamma \cdot a : T) \setminus \text{chan}(\Gamma) \cdot \bar{x}_i : \bar{T}_i \vdash_{\Theta} P_i : \emptyset\) and deduce that \(\Gamma \setminus \text{chan}(\Gamma) \cdot \bar{x}_i : \bar{T}_i \vdash_{\Theta} P_i : \emptyset\) is derivable. This is possible since \(a \notin \text{fn}(P_i)\) and \(\Gamma \vdash_{\Theta \setminus x} \phi\).

Next we construct the derivation

\[ \frac{\Gamma \cdot a : T \vdash_{\Theta} Q : e \quad \Gamma \vdash_{\Theta} \phi}{\Gamma \vdash_{\Theta} (\nu a : T)Q : e} \]  

Type NRes

and thus

\[ \frac{\Gamma \setminus \text{chan}(\Gamma) \cdot \bar{x}_i : \bar{T}_i \vdash_{\Theta} P_i : \emptyset \quad \text{fn}(\bar{T}_i) \cap \text{dom}(\Gamma) = \emptyset \quad \Theta(X_i) = (\bar{x}_i : \bar{T}_i) \quad \Gamma \vdash_{\Theta} (\nu a : T)Q : e}{\Gamma \vdash_{\Theta \setminus x} \text{def } D \text{ in } (\nu a : T)Q : e} \]  

Type

Note that the fact that \(\Gamma \vdash_{\Theta} \phi\) follows from \(\Gamma \vdash_{\Theta \setminus x} \phi\) is because the correct formation of \(\Gamma\) and \(\Theta\) may be determined separately.

(ii) For the second item the derivation must take the form:

\[ \frac{\Gamma \setminus \text{chan}(\Gamma) \cdot \bar{x}_i : \bar{T}_i \vdash_{\Theta} P_i : \emptyset \quad \text{fn}(\bar{T}_i) \cap \text{dom}(\Gamma) = \emptyset \quad \Theta(X_i) = (\bar{x}_i : \bar{T}_i) \quad \Gamma \vdash_{\Theta} (\nu a : T)Q : e}{\Gamma \vdash_{\Theta \setminus x} \text{def } D \text{ in } (\nu a : T)Q : e} \]  

Type

where the top right-hand judgement is derived as follows:

\[ \frac{\Gamma \cdot a : T \vdash_{\Theta} Q : e \quad \Gamma \vdash_{\Theta} \phi}{\Gamma \vdash_{\Theta} (\nu a : T)Q : e} \]  

Type NRes

From each \(\Gamma \setminus \text{chan}(\Gamma) \cdot \bar{x}_i : \bar{T}_i \vdash_{\Theta} P_i : \emptyset\) and the Weakening Lemma (Lemma 2.5) we deduce \(\Gamma \setminus \text{chan}(\Gamma) \cdot a : T \cdot \bar{x}_i : \bar{T}_i \vdash_{\Theta} P_i : \emptyset\). Finally, if we let \(\Gamma' = \Gamma \cdot a : T\), then we construct the derivation (note that \(\Gamma \setminus \text{chan}(\Gamma) \cdot a : T = \Gamma' \setminus \text{chan}(\Gamma')\)):

\[ \frac{\Gamma' \vdash_{\Theta \setminus x} \text{def } D \text{ in } Q : e \quad \Gamma \vdash_{\Theta \setminus x} \phi}{\Gamma \vdash_{\Theta \setminus x} (\nu a : T)\text{def } D \text{ in } Q : e} \]  

Type NRes

where the top left-hand judgement is derived as follows:

\[ \frac{\Gamma' \setminus \text{chan}(\Gamma') \cdot \bar{x}_i : \bar{T}_i \vdash_{\Theta} P_i : \emptyset \quad \text{fn}(\bar{T}_i) \cap \text{dom}(\Gamma) = \emptyset \quad \Theta(X_i) = (\bar{x}_i : \bar{T}_i) \quad \Gamma' \vdash_{\Theta} Q : e}{\Gamma' \vdash_{\Theta \setminus x} \text{def } D \text{ in } Q : e} \]  

Type Def

**SC Def Par.** (i) For the first item the derivation must end in:

\[ \frac{\Gamma \vdash_{\Theta \setminus x} \text{def } D \text{ in } P : e_P \quad \Delta \vdash_{\Theta \setminus x} Q : e_Q}{\Gamma \circ \Delta \vdash_{\Theta \setminus x} (\text{def } D \text{ in } P)Q : e_P + e_Q} \]  

Type Par

30
where \( \Gamma \prec \Delta \) and the top left-hand judgement is derived as follows:

\[
\Gamma \vdash \Theta \{ \cdot \} \quad \text{def} D \quad \text{in} \quad P : e_P
\]

Then if we let \( \Upsilon = \Gamma \circ \Delta \), we construct the derivation

\[
\Upsilon \vdash \Theta \{ \cdot \} \quad \text{def} D \quad \text{in} \quad P : e_P + e_Q
\]

where the top right-hand judgment is derived as follows:

\[
\Gamma \vdash \Theta \{ \cdot \} \quad \text{in} \quad P : e_P \quad \Delta \vdash \Theta \{ \cdot \} \quad Q : e_Q
\]

\[
\Upsilon \vdash \Theta \{ \cdot \} \quad \text{in} \quad P \mid Q : e_P + e_Q
\]

Note that we have used the fact that \( \Delta \vdash \Theta \{ \cdot \} \quad Q : e_Q \) implies \( \Delta \vdash \Theta \{ \cdot \} \quad Q : e_Q \). Also, we have used the trivial result that since \( \Gamma \prec \Delta \), then \( \Upsilon \vdash \Theta \{ \cdot \} = \Gamma \vdash \Theta \{ \cdot \} \).

(ii) For the second item we proceed along the lines of the previous one.

**SC Def And.** In both items, the desired derivation results from a straightforward regrouping of subderivations.

\[\square\]

**Lemma B.9 (Substitution Lemma)** If \( \Gamma \vdash x : T \cdot \Delta \vdash \Theta \{ \cdot \} \) and \( \Sigma \vdash v : T \) and \( x \notin \text{fn}(\Gamma) \) and \( \Gamma \prec \Sigma \), then \( \Gamma \vdash \Delta \{ x \leftarrow v \} \vdash \Theta \{ x \leftarrow v \} \).

**Proof.** By induction on the length of the derivation of \( \Gamma \vdash x : T \cdot \Delta \vdash \Theta \{ \cdot \} \). \(\square\)

**Corollary B.10 (Substitution Corollary)** Suppose \( \Gamma \vdash (v) : (\vec{x} : \vec{T}) \). If \( \Gamma \vdash \vec{x} : \vec{T} \cdot \Delta \vdash \Theta \{ \cdot \} \) with \( \vec{x} \notin \Gamma \), then \( \Gamma \vdash \Delta \{ \vec{x} \leftarrow \vec{v} \} \vdash \Theta \{ \vec{x} \leftarrow \vec{v} \} \).

**Proof.** By induction on \( n \).

* \( n = 1 \). Then we have \( \Gamma \vdash x_1 : T_1 \cdot \Delta \vdash \Theta \{ \cdot \} \) and \( \Gamma \vdash v : T_1 \). We apply the Substitution Lemma (Lemma B.9) and conclude with \( \Gamma \vdash \Delta \{ x_1 \leftarrow v_1 \} \vdash \Theta \{ x_1 \leftarrow v_1 \} \).

* \( n > 1 \). We take \( \Delta = x_n : T_n \cdot \Delta \) and apply the induction hypothesis noting that \( \Gamma \vdash (v) : (\vec{x} : \vec{T}) \) implies \( \Gamma \vdash (v') : (\vec{x}' : \vec{T}' \cdot \vec{v}') \) where \( v', \vec{x}' \) and \( \vec{T}' \) result from \( v, \vec{x} \) and \( \vec{T} \), respect. by eliminating the final element \( v_n, x_n \) and \( T_n \), respect. As a result we obtain:

\[
B.1 \quad \Gamma, x_n : T, \Delta \{ x' \leftarrow v' \} \vdash \Theta \{ x' \leftarrow v' \}
\]

Now from the hypothesis \( \Gamma \vdash (v) : (\vec{x} : \vec{T}) \) we know that

\[
B.2 \quad \Gamma \vdash v_n : T \{ x' \leftarrow v' \}
\]

Finally, we apply the Substitution Lemma to (B.1) and (B.2) and conclude. \(\square\)
B.1 Subject Reduction and Safety

Lemma B.11 (≡-elimination) If \( P \xrightarrow{\nu} P' \), then for some \( Q \equiv P \) and \( Q' \equiv P' \), \( Q \xrightarrow{\nu} Q' \) is derivable without using the rule \( \text{Trans} \equiv \).

Proof. By induction on the derivation of \( P \xrightarrow{\nu} P' \). The inductive cases are:

- Trans Par. In this case the derivation ends

\[
\frac{P_1 \xrightarrow{\nu} P'_1}{P_1 | P_2 \xrightarrow{\nu} P'_1 | P_2} \quad \text{Trans Par}
\]

By the i.h. there exists \( Q_1 \equiv P_1 \) and \( Q'_1 \equiv P'_1 \) s.t. \( Q_1 \xrightarrow{\nu} Q'_1 \) is derivable without using the rule \( \text{Trans} \equiv \). We may conclude by taking \( Q = Q_1 | P_2 \) and \( Q' = Q'_1 | P_2 \).

- Trans Def2. In this case the derivation ends

\[
\frac{P_1 \xrightarrow{\psi} P'_1}{\text{def } D \text{ in } P_1 \xrightarrow{\psi} \text{def } D \text{ in } P'_1} \quad \text{Trans Def2}
\]

By the i.h. there exists \( Q_1 \equiv P_1 \) and \( Q'_1 \equiv P'_1 \) s.t. \( Q_1 \xrightarrow{\nu} Q'_1 \) is derivable without using the rule \( \text{Trans} \equiv \). We may conclude by taking \( Q = \text{def } D \text{ in } Q_1 \) and \( Q' = \text{def } D \text{ in } Q'_1 \).

- Trans ≡. In this case the derivation ends

\[
\frac{P \equiv R \quad R \xrightarrow{\nu} R' \quad R' \equiv P'}{P \xrightarrow{\nu} P'} \quad \text{Trans ≡}
\]

By the i.h. there exists \( Q \equiv R \) and \( Q' \equiv R' \) s.t. \( Q \xrightarrow{\nu} Q' \) is derivable without using the rule \( \text{Trans} \equiv \). We may conclude since \( Q \equiv R \equiv P \) and \( Q' \equiv R' \equiv P' \).

\[ \square \]

Proof. [Of Subject Reduction]

(i) There are five cases to consider:

(a) If \( P \xrightarrow{\tau} P' \) derives from the \( \text{Trans} \) Link transition, then by ≡-Elimination (Lemma B.11):

\[
P \equiv \text{accept } a(k) \text{ in } P_1 | \text{request } a(k) \text{ in } P_2 | R
\]

\[
P' \equiv (\nu k)P_1 | P_2 | R
\]

By Subsumption Elimination (Lemma B.3) and Subject Congruence the derivation of \( \Gamma \vdash \sigma P : e \) takes the following form:

- On the one hand, we have:

\[
\begin{align*}
\Gamma \cdot a : \sigma(a) \cdot k : a & \vdash_0 P_1 : e_{P_1} & \text{Type Acpt} & \Delta \cdot a : \sigma(a) \cdot k : \tau & \vdash_0 P_2 : e_{P_2} \\
(\Gamma \circ \Delta) \cdot a : \sigma(a) & \vdash_0 \text{accept } a(k) \text{ in } P_1 : e_{P_1} & \Delta \cdot a : \sigma(a) & \vdash_0 \text{request } a(k) \text{ in } P_2 : e_{P_2}
\end{align*}
\]

32
• On the other, the derivation must end with an application of the
  Type Par rule, as indicated below:

\[
\frac{(\Gamma \circ \Delta) \cdot a : \sigma(\alpha) \vdash \text{accept} \ a(k) \ \text{in} \ P_1 | \text{request} \ a(k) \ \text{in} \ P_2 : e_{P_1} + e_{P_2} \quad \Sigma \cdot a : \sigma(\alpha) \vdash R}{((\Gamma \circ \Delta) \circ \Sigma) \cdot a : \sigma(\alpha) \vdash \text{accept} \ a(k) \ \text{in} \ P_1 | \text{request} \ a(k) \ \text{in} \ P_2 : R : (e_{P_1} + e_{P_2}) + e_R}
\]

where

\[
\Gamma = ((\Gamma \circ \Delta) \circ \Sigma) \cdot a : \sigma(\alpha)
\]

\[
\Gamma \triangleleft \Delta \quad \Gamma \circ \Delta \triangleleft \Sigma \quad (e_{P_1} + e_{P_2}) + e_R \leq e
\]

We may then derive \( \Gamma \vdash_\Theta (\nu k)(P_1 | P_2) \vdash e \) as follows:

\[
\frac{\Gamma \cdot a : \sigma(\alpha) \cdot k : \alpha \vdash_\Theta P_1 : e_{P_1} \quad \Delta \cdot a : \sigma(\alpha) \cdot k : \overline{\sigma} \vdash_\Theta P_2 : e_{P_2}}{((\Gamma \circ \Delta) \cdot a : \sigma(\alpha) \vdash_\Theta (\nu k)(P_1 | P_2) : e_{P_1} + e_{P_2}) \quad \Sigma \cdot a : \sigma(\alpha) \vdash_\Theta R}{(\Gamma \circ \Delta) \cdot a : \sigma(\alpha) \vdash_\Theta (\nu k)(P_1 | P_2) : e_{P_1} + e_{P_2} \quad \Sigma \cdot a : \sigma(\alpha) \vdash_\Theta R : (e_{P_1} + e_{P_2} + e_R)}
\]

where \( f = \text{fnMult}(\alpha) \).

Finally, we conclude by Subject Concurrency.

(b) If \( P \xrightarrow{T} P' \) derives from the Trans Comm transition, then by \( \equiv \)-Elimination (Lemma [B.11]):

\[
P \equiv (k!v; P_1) | (k?y \text{in} P_2) \vdash R
\]

\[
P' \equiv P_1 | P_2\{y \leftarrow v\} \vdash R
\]

By Subsumption-Elimination (Lemma [B.3]) and Subject Concurrency the derivation of \( \Gamma \vdash_\Theta P : e \) takes the following form:

• On the other hand, we have:

\[
\frac{\Gamma \vdash_\Theta v : T \quad \Gamma \cdot k : \alpha \vdash_\Theta P_1 : e_{P_1} \quad \Gamma \cdot k \vdash_\Theta [y : T' \alpha \vdash_\Theta \circ {k} \vdash_\Theta P_1 : e_{P_1} + e' \{y \leftarrow v\}]}{\Delta \cdot y : T \cdot k : \overline{\alpha} \vdash_\Theta P_2 : e_{P_2} \quad \text{fn}(e_{P_2} - e') \subseteq \text{dom}(\Delta) \quad \Delta \cdot k \vdash_\Theta [y : T' \alpha \vdash_\Theta \circ {k} \vdash_\Theta P_2 : e_{P_2} - e']}{\text{Type Snd}}
\]

• On the other hand, we have:

\[
\frac{\Delta \cdot y : T \cdot k : \overline{\alpha} \vdash_\Theta P_2 : e_{P_2} \quad \text{fn}(e_{P_2} - e') \subseteq \text{dom}(\Delta) \quad \Delta \cdot k \vdash_\Theta [y : T' \alpha \vdash_\Theta \circ {k} \vdash_\Theta P_2 : e_{P_2} - e']}{\text{Type Snd}}
\]

• Finally, the derivation ends in an application of Type Par to the above two derivations yielding the top left-hand judgement in:

\[
\frac{(\Gamma \circ \Delta) \cdot k : \bot \vdash_\Theta (k!v; P_1) | (k?x \text{in} P_2) : e'' \quad \Sigma \vdash_\Theta R : e_R}{((\Gamma \circ \Delta) \cdot k : \bot \vdash_\Theta (k!v; P_1) | (k?x \text{in} P_2) : R : e'' + e_R)}{\text{Type Par}}
\]

where

\[
f = \text{fnMult}(\uparrow [y : T]e'; \alpha)
\]

\[
e'' = (e_{P_1} + e' \{y \leftarrow v\}) + (e_{P_2} - e')
\]

\[
(e_{P_1} + e' \{y \leftarrow v\}) + (e_{P_2} - e') + e_R \leq e
\]

\[
\Gamma \triangleleft \Delta \quad (\Gamma \circ \Delta) \cdot k : \bot \triangleleft \Sigma
\]

33
From $\Delta \cdot y : T \cdot k : \overline{\alpha} \vdash e_P : e_{P_2}$ and $\Gamma \vdash v : T$ and $\Delta \propto \Gamma$ and the Substitution Lemma, we deduce

$$\Delta \cdot k : \overline{\alpha} \{y \leftarrow v\} \vdash e_P \{y \leftarrow v\}$$

Next we construct the derivation:

$$\Gamma \cdot k : \alpha \{y \leftarrow v\} \vdash e_{P_1} \quad \Delta \cdot k : \overline{\alpha} \{y \leftarrow v\} \vdash e_{P_2} \{y \leftarrow v\} \quad \text{Type Par}$$

$$(\Gamma \circ \Delta) \cdot k : \perp \vdash e_{P_1} + e_{P_2} \{y \leftarrow v\} + e_R$$

where $f' = \text{fnMult}(\alpha \{y \leftarrow v\})$.

Finally, we introduce another application of Type Par:

$$((\Gamma \circ \Delta) \cdot k : \perp \vdash e_{P_1} \{y \leftarrow v\} + e_{P_2} \{y \leftarrow v\} + e_R \quad \Sigma \vdash R : e_R \quad \text{Type Par}$$

We are left to verify that $e_{P_1} + e_{P_2} \{y \leftarrow v\} + e_R \leq e$. We reason as follows:

$$e_{P_1} + e_{P_2} \{y \leftarrow v\} + e_R$$

$$= e_{P_1} + (e_{P_2} - e' + e') \{y \leftarrow v\} + e_R$$

$$= e_{P_1} + (e_{P_2} - e') \{y \leftarrow v\} + e' \{y \leftarrow v\} + e_R$$

$$= e_{P_1} + e_{P_2} - e + e' \{y \leftarrow v\} + e_R$$

$$\leq e$$

(hypothesis)

(c) If $P \overset{\tau}{\rightarrow} P'$ derives from the Trans Brach transition, then by $\equiv$ Elimination (Lemma B.11):

$$P \equiv (k < l_j; Q) \mid (k \triangleright \{l_i : Q_i\}) \mid R$$

$$P' \equiv Q \mid Q_j \mid R \quad j \in 1..n$$

By Subsumption-Elimination (Lemma B.3) and Subject Congruence the derivation of $\Gamma \vdash e : P$ takes the following form:

- On the one hand, we have the derivation

$$\Gamma \cdot k : \alpha \vdash Q : e_Q \quad 1 \leq j \leq n\quad \Gamma \cdot k : \oplus \{l_i : \alpha_i\} e' \vdash e \quad \text{Type Sel}$$

$$\Gamma \cdot k : \oplus \{l_i : \alpha_i\} e' \vdash k < l_j; Q : e_Q + e'$$

- On the other,

$$\Delta \cdot k \vdash Q_1 ; e_{Q_1} \quad \cdots \quad \Delta \cdot k \vdash \overline{\alpha} \vdash Q_n ; e_{Q_n} \quad \Delta \cdot k \vdash \oplus \{l_i : \alpha_i\} e' \vdash e \quad \text{Type Brach}$$

$$\Delta \cdot k \vdash \oplus \{l_i : \alpha_i\} e' \vdash \{l_i : Q_i\} ; (\bigvee e_{Q_i}) - e'$$

- From the above two derivations we obtain:

$$\Gamma \cdot k \vdash \oplus \{l_i : \alpha_i\} e' \vdash k < l_j; Q : e_Q + e' \quad \Delta \cdot k \vdash \oplus \{l_i : \alpha_i\} e' \vdash \{l_i : Q_i\} ; (\bigvee e_{Q_i}) - e' \quad \text{Type Brach}$$

$$(\Gamma \circ \Delta) \cdot k \vdash \perp \vdash (k < l_j; Q) \mid (k \triangleright \{l_i : Q_i\}) \vdash e_Q + \bigvee e_{Q_i}$$

where $f = \text{fnMult}(\oplus \{l_i : \alpha_i\} e')$.  

34
Finally, the derivation of $\Gamma \vdash Q : e$ must end with:

\[
(\Gamma \cup \Delta) \cdot k : \bot_f \vdash (k < l_j; Q)|\{i : Q_i\} : e_Q + \bigvee e_{Q_i} \quad \Sigma \vdash R : e_R \quad \text{Type Par}
\]

We then derive:

\[
(\Gamma \cup \Delta) \cdot k : \bot_f \vdash Q|\{i : Q_i\} : e_Q + e_{Q_j}
\]

\[
\Sigma \vdash R : e_R \quad \text{Type Par}
\]

where $f' = \text{fnMult}(\alpha_j)$. Finally, we conclude by using the rule Type Subsum (note that $e_Q + e_{Q_j} + e_R \leq e_Q + \bigvee e_{Q_i} + e_R$) and Subject Congruence.

(d) If $P \xrightarrow{\tau} P'$ derives from the Trans Catch transition, then by $\equiv$-Elimination (Lemma B.3.1):

\[
P \equiv (\text{throw } k[k']; Q_1)|\{\text{catch } k(k') \text{ in } Q_2\}|\ R
\]

\[
P' \equiv Q_1|\ Q_2\ R, \quad j \in 1..n
\]

By Subsumption-Elimination (Lemma B.3.3) and Subject Congruence the derivation of $\Gamma \vdash P : e$ takes the following form:

• On the one hand we have:

\[
\Gamma \cdot k : \alpha \vdash Q_1 : e_Q_1\quad \Gamma \cdot k' : \beta \cdot k : \uparrow [y : \beta]e'; \alpha \vdash \circ \quad \text{Type Thr}
\]

\[
\Gamma \cdot k' : \beta \cdot k : \uparrow [y : \beta]e'; \alpha \vdash \text{throw } k[k']; Q_1 : e_Q_1 + e'\{y \leftarrow k'\}
\]

• On the other we have:

\[
\Delta \cdot k' : \beta \cdot k : \uparrow [y : \beta]e'; \alpha \vdash \circ \quad \text{Type Cat}
\]

\[
\Delta \cdot k : \downarrow [y : \beta]e'; \alpha \vdash \text{catch } k(k') \text{ in } Q_2 : e_Q_2 - e'\{y \leftarrow k'\}
\]

• Finally, as a result of applying Type Par to the above two derivations we obtain the top left-hand judgement in:

\[
(\Gamma \cup \Delta) \cdot k' : \beta \cdot k : \bot_f \vdash \text{throw } k[k']; Q_1|\{\text{catch } k(k') \text{ in } Q_2\} : e'' \quad \Sigma \vdash R : e_R \quad \text{Type Par}
\]

\[
((\Gamma \cup \Delta) \cdot k' : \beta \cdot k : \bot_f) \circ \Sigma \vdash \text{throw } k[k']; Q_1|\{\text{catch } k(k') \text{ in } Q_2\}|\ R : e'' + e_R
\]

where

\[
f = \text{fnMult}(\uparrow [y : \beta]e'; \alpha)
\]

\[
e'' = (e_Q_1 + e'\{y \leftarrow k'\}) + (e_Q_2 - e'\{y \leftarrow k'\})
\]

\[
(e_Q_1 + e'\{y \leftarrow k'\}) + (e_Q_2 - e'\{y \leftarrow k'\}) + e_R \leq e
\]

\[
\Gamma \cdot k' : \beta \cdot k : \uparrow [y : \beta]e'; \alpha \times \Delta \cdot k : \downarrow [y : \beta]e'; \alpha \quad ((\Gamma \cup \Delta) \cdot k' : \beta \cdot k : \bot_f) \times \Sigma
\]

We proceed as follows:

\[
\Gamma \cdot k : \alpha \vdash Q_1 : e_Q_1\quad \Delta \cdot k' : \beta \cdot k : \uparrow [y : \beta]e'; \alpha \vdash Q_2 : e_Q_2
\]

\[
\Sigma \vdash R : e_R \quad \text{Type Par}
\]

\[
(\Gamma \cup \Delta) \cdot k' : \beta \cdot k : \bot_f \vdash Q_1|\ Q_2 : e_Q_1 + e_Q_2 \quad \Sigma \vdash R : e_R \quad \text{Type Par}
\]

\[
((\Gamma \cup \Delta) \cdot k' : \beta \cdot k : \bot_f) \circ \Sigma \vdash Q_1|\ Q_2\ R : e_Q_1 + e_Q_2 + e_R
\]
where \( f' = \text{fnMult} (\alpha) \). Note also that by hypothesis
\[
(e_{Q_1} + e' \{ y \leftarrow k' \}) + (e_{Q_2} - e' \{ y \leftarrow k' \}) + e_R
\]
\[
= e_{Q_1} + e_{Q_2} + e_R
\]
\[
\leq e
\]
(e) If \( P \xrightarrow{\tau} P' \) derives from the Trans Def1 transition, then by \( \equiv \)-Elimination (Lemma [B.11]):
\[
P \equiv \text{def } D \in (X_j[v] \mid Q) R
\]
\[
P' \equiv \text{def } D \in (P_j \{ x_j \leftarrow \bar{v} \} \mid Q) R, \quad X_j(x_j) = P_j \in D
\]
By Subsumption-Elimination (Lemma [B.3]) and Subject Congruence the derivation of \( \Gamma \vdash_\Theta P : e \) takes the following form:

- On the one hand, we have:
  \[
  \frac{\Gamma \vdash_\Theta (\bar{v} : T_j) \quad X_j : (x_j : T_j) \in \Theta \quad \text{ranCh}(\Gamma) \subseteq \{ 1, \bot_j \} \quad \text{Type PVar}}{\Gamma \vdash_\Theta X_j[\bar{v}] : [\|]}
  \quad \frac{\Delta \vdash_\Theta Q : e_Q}{\Gamma \circ \Delta \vdash_\Theta X_j[\bar{v}] \mid Q : e_Q}
  \]
  where \( \Gamma \asymp \Delta \).

- On the other hand, if we let \( \Upsilon = \Gamma \circ \Delta \), then we have:
  \[
  \Upsilon \setminus \text{chan}(\Upsilon) \cdot x_1 : \bar{T_1} \vdash_\Theta P_1 : ([\|] \ldots \Upsilon \setminus \text{chan}(\Upsilon) \cdot x_n : \bar{T_n} \vdash_\Theta P_n : [\|] \quad \Upsilon \vdash_\Theta X_j[\bar{v}] : e_Q
  \]
  where for each \( i \in 1..n \) we have \( \Theta(X_i) = (x_i : T_i) \) and \( \text{fn}(\bar{T_i}) \cap \text{dom}(\Upsilon) = \emptyset \).

We take the following steps.

Since \( \Gamma \asymp \Delta \), then \( (\Gamma \circ \Delta) \setminus \text{chan}(\Gamma \circ \Delta) = \Gamma \setminus \text{chan}(\Gamma) \). Thus we know that
\[
(\text{B.3}) \quad \Gamma \setminus \text{chan}(\Gamma) : x_j : T_j \vdash_\Theta P_j : [\|]
\]
is derivable.
Consider all the channel names \( \bar{t} \) in \( \text{dom}(\Gamma) \). Note that by the condition \( \text{ranCh}(\Gamma) \subseteq \{ 1, \bot_j \} \) we know that \( \Gamma(\bar{t}) \subseteq \{ 1, \bot_j \} \). We may thus apply Weakening (Lemma [B.6]) to (B.3) and obtain
\[
(\text{B.4}) \quad \Gamma \cdot x_j : T_j \vdash_\Theta P_j : [\|]
\]
We are now in conditions of applying the Substitution Corollary (Corollary [B.10]). It is applied to (B.4) knowing \( \Gamma \vdash_\Theta (\bar{v}) : (x : T_j) \).

As a result we obtain
\[
(\text{B.5}) \quad \Gamma \vdash_\Theta P_j \{ x_j \leftarrow \bar{v} \} : [\|]
\]
Finally, we construct the following derivation, where the top left-hand judgement is (B.3):
\[
\frac{\Gamma \vdash_\Theta P_j \{ x_j \leftarrow \bar{v} \} : [\|] \quad \Delta \vdash_\Theta Q : e_Q \quad \Gamma \asymp \Delta}{\Gamma \circ \Delta \vdash_\Theta P_j \{ x_j \leftarrow \bar{v} \} \mid Q : e_Q}
\]
Type Par
36
We conclude by an application of the Type Def rule.

(f) If \( P \xrightarrow{\begin{smallmatrix} \text{begin} \\ L \end{smallmatrix}} P' \), then by the \( \equiv \)-Elimination (Lemma B.11):

\[
P \equiv \text{begin } L; Q \parallel R
\]

\[
P' \equiv Q \parallel R
\]

By Subsumption-Elimination (Lemma B.3) and Subject Concurrency the derivation of \( \Gamma \vdash e : e \) takes the following form:

\[
\frac{\Gamma \vdash Q : e_Q \quad \text{fn}(L) \subseteq \text{dom}(\Gamma)}{\Gamma \vdash \text{begin } L; Q : e_Q - (|L|)} \quad \text{Type Bgn}
\]

\[
\frac{\Delta \vdash e_R \quad \Gamma \preceq \Delta}{\Gamma \circ \Delta \vdash \text{begin } L; Q : e_Q - (|L|) + e_R} \quad \text{Type Par}
\]

We then derive:

\[
\frac{\Gamma \vdash Q : e_Q \quad \Delta \vdash e_R}{\Gamma \circ \Delta \vdash Q : e_Q + e_R} \quad \text{Type Par}
\]

Since \( e_Q - (|L|) + e_R \leq e \), then \( e_Q + e_R \leq e + (|L|) \). Finally, we conclude by Subject Concurrency.

(g) If \( P \xrightarrow{\begin{smallmatrix} \text{end} \\ L \end{smallmatrix}} P' \), then by the \( \equiv \)-Elimination (Lemma B.11):

\[
P \equiv \text{end } L; Q \parallel R
\]

\[
P' \equiv Q \parallel R
\]

By Subsumption-Elimination (Lemma B.3) and Subject Concurrency the derivation of \( \Gamma \vdash e : e \) takes the following form:

\[
\frac{\Gamma \vdash Q : e_Q \quad \text{fn}(L) \subseteq \text{dom}(\Gamma)}{\Gamma \vdash \text{end } L; Q : e_Q + (|L|)} \quad \text{Type End}
\]

\[
\frac{\Delta \vdash e_R \quad \Gamma \preceq \Delta}{\Gamma \circ \Delta \vdash \text{end } L; Q : e_Q + (|L|) + e_R} \quad \text{Type Par}
\]

We then derive:

\[
\frac{\Gamma \vdash Q : e_Q \quad \Delta \vdash e_R}{\Gamma \circ \Delta \vdash Q : e_Q + e_R} \quad \text{Type Par}
\]

Since \( e_Q + (|L|) + e_R \leq e \) (note that from this we deduce that \( (|L|) \in e \)), then \( e_Q + e_R \leq e - (|L|) \). Finally, we conclude by Subject Concurrency.

(h) If \( P \xrightarrow{\text{res}(a : T)} P' \), then by \( \equiv \)-Elimination (Lemma B.11):

\[
P \equiv (\nu a : T)Q
\]

\[
P' \equiv Q
\]

By Subsumption-Elimination (Lemma B.3) and Subject Concurrency the derivation of \( \Gamma \vdash e : e \) takes the following form:

\[
\frac{\Gamma \cdot a : T \vdash Q : e_Q \quad \Gamma \vdash e \quad \Gamma \vdash e \quad \text{Type NRes}}{\Gamma \vdash (\nu a : T)Q : e_Q}
\]

37
where \( e_Q \leq e \). By Subject Congruence and an application of the Type Subsum rule we obtain the desired result.

(i) If \( P \xrightarrow{\text{res}(k)} P' \), then by \( \equiv \)-Elimination (Lemma B.11):

\[
P \equiv (\nu k)Q
\]

\[
P' \equiv Q
\]

By Subsumption-Elimination (Lemma B.3) and Subject Congruence the derivation of \( \Gamma \vdash \Theta P : e \) takes the following form:

\[
\Gamma \cdot k : \bot \vdash \Theta Q : e_Q
\]

\[
\Gamma \vdash \Theta (\nu k)Q : e_Q
\]

where \( e_Q \leq e \). By Subject Congruence and an application of the Type Subsum rule we obtain the desired result.

(j) If \( P \overset{\text{def}(\vec{x} : (\vec{x}_i : \bar{T}_i))}{\rightarrow} P' \), then by \( \equiv \)-Elimination (Lemma B.11):

\[
P \equiv \text{def } D \text{ in } Q
\]

\[
P' \equiv Q
\]

where \( D = X_1(\vec{x}_1^\ast) = P_1 \ldots \text{and } \ldots X_n(\vec{x}_n^\ast) = P_1 \) for some process terms \( P_1, \ldots, P_n \). By Subsumption-Elimination (Lemma B.3) and Subject Congruence the derivation of \( \Gamma \vdash \Theta P : e \) takes the following form:

\[
\Gamma \setminus \text{chan}(\Gamma) \cdot \vec{x}_1 : \bar{T}_1 \vdash \Theta P_1 : (\|) \ldots \Gamma \setminus \text{chan}(\Gamma) \cdot \vec{x}_n : \bar{T}_n \vdash \Theta P_n : (\|) \Gamma \vdash \Theta Q : e_Q
\]

\[
\Gamma \vdash \Theta \setminus \vec{x} \text{ def } X_1(\vec{x}_1^\ast) = P_1 \ldots \text{and } \ldots X_n(\vec{x}_n^\ast) = P_n \text{ in } Q : e_Q
\]

where \( e_Q \leq e \). We take the derivation of \( \Gamma \vdash \Theta Q : e_Q \) and an application of the Type Subsum rule and obtain the desired result.

\[ \square \]

Lemma B.12 If \( \Gamma \vdash \Theta P : e \) and \( P \overset{s}{\rightarrow} P' \) and \( \text{gn}(s) \cap \text{dom}(\Gamma) = \emptyset \), then \( \text{ends}(s) \leq \text{begins}(s) + e \).

**Proof.** Induction on the derivation of \( P \overset{s}{\rightarrow} P' \) (Definition 3.1).

(i) \( P \equiv P' \). Then the result follows from Subject Congruence.

(ii) \( s = \nu s' \) with \( P \overset{\nu}{\rightarrow} P'' \) and \( P'' \overset{s'}{\rightarrow} P' \). We consider each possible case for \( \nu \).

- \( \nu = \tau \). By Subject Reduction \( \Gamma' \vdash \Theta P'' : e \) where \( \Gamma' \) (possibly) differs from \( \Gamma \) in the effects assigned to some occurrences of \( \bot \). By induction hypothesis \( \text{ends}(s') \leq \text{begins}(s') + e \) and since \( \text{ends}(\tau) = \text{begins}(\tau) = (\|) \) we may conclude.

- \( \nu = \text{begin } L \) with \( \text{fn}(L) \cap \text{gn}(s') = \emptyset \). By Subject Reduction \( \Gamma \vdash \Theta P'' : e + (\| L) \) and by the induction hypothesis

\[
\text{ends}(s') \leq \text{begins}(s') + e + (\| L)
\]

38
Thus
\[
\text{ends}(s) = \text{ends}(s') \\
\leq \text{begins}(s') + e + \|L\| \\
= \text{begins}(s) + e
\]

• \(v = \text{end} L\) with \(\text{fn}(L) \cap \text{gn}(s') = \emptyset\). By Subject Reduction \(\Gamma \vdash_\emptyset P'' : e - \|L\|\) with \(L \in e\) and by the induction hypothesis
\[
\text{ends}(s') \leq \text{begins}(s') + e - \|L\|
\]
Thus
\[
\text{ends}(s) = \text{ends}(s') + \|L\| \\
\leq \text{begins}(s') + e \\
= \text{begins}(s) + e
\]

• \(v = \text{res}(a : T)\) with \(\{a\} \cap \text{gn}(s') = \emptyset\). By Subject Reduction \(\Gamma \cdot a : T \vdash_\emptyset P'' : e\) and by the induction hypothesis
\[
\text{ends}(s') \leq \text{begins}(s') + e
\]
Thus
\[
\text{ends}(s) \leq \text{begins}(s) + e
\]

• \(v = \text{res}(k)\) with \(\{k\} \cap \text{gn}(s') = \emptyset\). Similar to the previous case.

\(\square\)

\textbf{Proof.} \cite{Bonelli:2015} - Safety | As in \cite{IANN10}. We repeat the argument for the sake of clarity. Suppose \(P \xrightarrow{s} P'\) for some trace \(s\) and process \(P'\). Without loss of generality we may assume that \(\text{gn}(s) \cap \text{dom}(\Gamma) = \emptyset\) (for we could otherwise rename names to our convenience). By Lemma 4.12 we have \(\text{ends}(s) \leq \text{begins}(s) + \|\|\), i.e. \(\text{ends}(s) \leq \text{begins}(s)\) as required.

\(\square\)