Minimal Session Types

Bachelor Thesis

Erik Voogd, University of Groningen

First supervisor
Dr. J.A. Perez Parra
University of Groningen

Second supervisor
Prof. dr. G.R. Renardel de Lavalette
University of Groningen

September 21, 2018

Abstract

Session types were introduced in the theory of types to formally specify and verify that communication in distributed systems follows the intended protocols. They were originally developed within the context of a high-level abstract language called the π-calculus, but they are finding their way to implementations in real programming languages. Processes modeled in the π-calculus can be broken down into minimal parallel components, while preserving a well-defined operational correspondence. In this thesis, we explore a way of defining this breakdown function on processes such that it also induces minimal session types. The main result is that the function preserves typability, and we conjecture that an operational correspondence holds.

Contents

1 Introduction

The world nowadays contains countless systems consisting of processes that are in a state of constant communication. In many such systems it is crucial that they behave well, which means that no runtime errors can occur. In a banking system, for example, the transferred money should end up on the intended bank account, and not get lost.

To ensure safe communication in distributed systems, we verify that programs follow the intended *protocols*. Protocols express what types of things are sent or received, and in what sequence they should be communicated. The theory of *session types* [2, 3, 10] allows us to formally specify protocols. Session types can be seen as types like `int` or `string`, but for communicating channels in programs, rather than for variables. For example, a session type could specify that a channel sends an `int`, then receives a `bool`, and then ends the session. Verifying correctness in a type system that implements session types implies that all communication follows the intended protocols. This would then prevent that transactions in banking systems are done twice, for example, or that the bank system and the user program are both waiting for input.

Session types find their roots within the context of the π-*calculus* [5, 9], which is a high-level abstract programming language designed to model concurrent, interactive processes. Over the last decade, session types have also found their way into implementations of mainstream programming languages [6, 13, 11].

Earlier research by Parrow [7] has shown how to model processes in an untyped π-calculus using only minimal parallel components, called *trios*. A trio is a process consisting of at most three actions: the first activates the process, the second is typically the elementary action, and the third activates the next trio. In this way, trios running in parallel are able to simulate sequential behaviour. Indeed, a main result by Parrow is that *concerts*, processes containing only trios, inherit all the expressive power of the untyped π-calculus, up to *weak bisimulation*, which describes a behavioural correspondence between processes.

To our knowledge, trios have not been researched within the context of session types. If processes can be modeled with minimal parallel components, it is easily conjectured that it can be designed in such a way that the components implement minimal protocols. The research question of this thesis is therefore: can we decompose standard session types into *minimal session types*, while preserving typing and establishing operational correspondence? There are examples of similar results that hold for untyped, but not for typed processes, so an answer to this question is far from trivial.

In this article, we use a π-calculus with a type system introduced by Vasconcelos [12], and we define a function on typed processes that induces minimal session types. Our main result is then that if a process is typable, then its trio breakdown is also typable. Since processes are assumed to be closed in this type system, we define a specialized weak bisimulation, and conjecture that a typed process is weakly bisimilar to its trio breakdown.

In Section 2, we present the syntax and semantics of the π-calculus. Then, in Section 3, we introduce types. After that, in Section 4, some useful definitions are introduced to work towards our main definition, which is presented in Section 5. Finally, the result is shown in Section 6 and in Section 7 we discuss the things that further research should cover.

$P ::=$		**Processes**:
	$\mathbf{0}$	termination
	$\overline{x}\langle v\rangle \,.\, P$	output
	$x(x) \,.\, P$	input
	$P \mid P$	parallel composition
	$(\boldsymbol{\nu} xx)P$	restriction
	$\mathsf{un}\ x(x) \,.\, P$	replication
$v ::=$		**Values**:
	x	name, variable
	$\mathsf{true} \mid \mathsf{false}$	basic type values

Figure 1: Process syntax rules

2 The Pi Calculus

The π-calculus is a process calculus that allows us to model processes $P, Q, \ldots$ that perform communication and that we can compose in parallel. The language that we use here is as described by Vasconcelos [12], but without replication. In this section we introduce the syntax and the operational semantics of the calculus. The reader will be familiarized with the language by use of an example.

2.1 Syntax

First of all, a terminated process, also referred to as *inaction*, is modeled by writing $\mathbf{0}$. This rule is the first one that is described in Figure 1, where all rules for the syntax of processes can be found.

The sequential operator " . " allows us to model successive actions. Actions of communication are ranged over by $\alpha, \beta, \ldots$, and $\alpha \,.\, P$ models a process that performs the action α, and then continues as process P. To model variables and channels, we use names, also called variables, from a countable set that is ranged over by $a, b, \ldots, x, y, \ldots$. Each communication action α can then be either output, written $\overline{a}\langle b\rangle$, or input, written $x(y)$. We write $\overline{a}\langle b\rangle$ to mean that channel a sends the variable b, and $x(y)$ means that channel x can receive an input that is then written to the new variable y. Here, names a and x are the *subjects* of communication, and b and y are referred to as the *objects*. Communicating nothing is written as $\overline{x}\langle\rangle$ or $y()$.

In the process $a(b) \,.\, P$, the name b is instantiated after the input on channel a, and so we say that b is a *bound* name in the process $a(b) \,.\, P$. Contrarily, the name a occurs *free* in $a(b) \,.\, P$.

Notation 1. The set of free names in a process P is denoted by $\mathtt{fn}(P)$, and $\mathtt{bn}(P)$ denotes the set of bound names in P. The same notation is used for prefixes α.

The π-calculus also allows us to model concurrency using the parallel operator " | ", and $P \mid Q$ denotes the parallel composition of the two processes P and Q. Note that we can write $\alpha \,.\, (P \mid Q)$,

whereas $P \,.\, (Q \mid R)$ is not a process generated by the rules in Figure 1. If a process is free of the parallel operator, it may also be referred to as a *thread*.

New names can be created with the restriction operator, "$\boldsymbol{\nu}$", which is pronounced as *new*. In our case, scope restriction is done on covariables, meaning that we create a pair of channels that are supposed to communicate with each other, or *synchronize*. The names x and y occur bound in $(\boldsymbol{\nu} xy)P$, and this models a process where x can synchronize with y inside the process P.

Persistency of processes is useful to model servers. Within the π-calculus, it is referred to as *replication* of a process. We stick to the convention and we annotate an input process with the un-predicate (see also Section 3) to model replication. Strictly speaking, the non-replicated input process is annotated with the lin-predicate, but is omitted, like in [12]. Intuitively, communication with a replicated process $\mathsf{un}\ x(y) \,.\, P$ produces a copy of P in parallel, while the original process continues to exist. This behavior will be made formal when introducing the semantics.

Every process that we can model is a term, and any term occurring in P, including P itself, is a subterm of P. For example, the number of subterms in $P \mid Q$ is one plus the number of subterms in both P and Q.

Notation 2. The number of subterms in a process P is denoted by $|P|$.

Example 1. To model a process Q that outputs the boolean value true, then inputs a variable, and then continues as a process Q', we can write

$$Q \triangleq \overline{x}\langle \mathsf{true} \rangle \,.\, x(v) \,.\, Q'$$

We can model another process that persistently receives a name through a channel, then sends the same name back, and finally terminates, by writing

$$R \triangleq \mathsf{un}\ y(v) \,.\, \overline{y}\langle v \rangle \,.\, \mathbf{0}.$$

Here, we have $\mathtt{bn}(R) = \{v\}$ and $\mathtt{fn}(R) = \{y\}$. If we want these processes to communicate, we compose them in parallel, and we can use the restriction operator to say that x and y were created to communicate with each other.

$$P \triangleq (\boldsymbol{\nu} xy)(Q \mid R) = (\boldsymbol{\nu} xy)(\overline{x}\langle \mathsf{true} \rangle \,.\, x(v) \,.\, Q' \mid \mathsf{un}\ y(v) \,.\, \overline{y}\langle v \rangle \,.\, \mathbf{0}).$$

In this process P, the name v is bound by an input twice, but their scope is different. Note that the choice of the name for one of the two bound variables v is arbitrary. For example, by α*-conversion* we have

$$\mathsf{un}\ y(v) \,.\, \overline{y}\langle v \rangle \,.\, \mathbf{0} = \mathsf{un}\ y(a) \,.\, \overline{y}\langle a \rangle \,.\, \mathbf{0}$$

Last, the reader should verify that $|P| = 7 + |Q'|$.

$$P \mid \mathbf{0} \equiv P \qquad P \mid Q \equiv Q \mid P \qquad (P \mid Q) \mid R \equiv P \mid (Q \mid R)$$

$$(\boldsymbol{\nu} wx)(\boldsymbol{\nu} yz)P \equiv (\boldsymbol{\nu} yz)(\boldsymbol{\nu} wx)P \qquad (\boldsymbol{\nu} xy)\mathbf{0} \equiv \mathbf{0} \qquad (\boldsymbol{\nu} xy)P \mid Q \equiv (\boldsymbol{\nu} xy)(P \mid Q)$$

Figure 2: Axioms defining a structural congruence relation on processes

$$(\boldsymbol{\nu} xy)(\overline{x}\langle z\rangle \,.\, P \mid \mathsf{lin}\; y(w) \,.\, Q \mid R) \to (\boldsymbol{\nu} xy)(P \mid Q\{^z/_w\} \mid R) \qquad [R\text{-}LinCom]$$

$$(\boldsymbol{\nu} xy)(\overline{x}\langle z\rangle \,.\, P \mid \mathsf{un}\; y(w) \,.\, Q \mid R) \to (\boldsymbol{\nu} xy)(P \mid Q\{^z/_w\} \mid \mathsf{un}\; y(w) \,.\, Q \mid R) \qquad [R\text{-}UnCom]$$

$$\frac{P \to Q}{(\boldsymbol{\nu} xy)P \to (\boldsymbol{\nu} xy)Q} \quad [R\text{-}Res] \qquad \frac{P \to Q}{P \mid R \to Q \mid R} \quad [R\text{-}Par]$$

$$\frac{P' \equiv P \qquad P \to Q \qquad Q \equiv Q'}{P' \to Q'} \qquad [R\text{-}Struct]$$

Figure 3: Set of reduction rules defining operational semantics

2.2 Semantics

So far, we have only shown how we can model processes. For the formal semantics, we will introduce a set of rules that describe how the syntax of processes can be reduced. Before we do this, we use a set of axioms to define structural congruence of processes.

Structural congruence of processes is defined to be the smallest congruence relation that respects the axioms in Figure 2. Being a congruence relation means that it is an equivalence relation, for which reflexivity, symmetry, and transitivity hold. It furthermore means that all operators, sequential, parallel, and restriction, are preserved by it. For example, if $P \equiv Q$, then $\alpha \,.\, P \equiv \alpha \,.\, Q$. Being the smallest such congruence relation intuitively means that structural congruence holds *only if* it is by these axioms.

The first axiom says that the operator for parallel composition has an identity element, namely the terminated process. The next two rules say that parallel composition is defined to be both commutative and associative. The restriction operator is also commutative by the fourth axiom, and by the fifth, restrictions on the terminated process can be removed.

The last axiom says that a scope of a pair of covariables can be extended, called scope extrusion, or made smaller. To see why this axiom is needed, the reader is referred to Example 3. For scope extrusion, which is applying the relation from left to right, we use the standard variable convention by Barendregt [1], which originated in the λ-calculus. If the relation is applied from right to left, we naturally need that $x, y \notin \mathtt{fn}(Q)$.

The operational semantics are defined by a set of reduction rules (see Figure 3). Processes in the π-calculus that we work with are in a closed environment, which means that they do not interact with the outside world. Reduction can then only happen if an output and an input are done simul-

taneously in different parallel processes on two channels under one scope restriction, as in:

$$(\boldsymbol{\nu} xy)(\overline{x}\langle z\rangle \,.\, P \mid y(w) \,.\, Q \mid R) \rightarrow (\boldsymbol{\nu} xy)(P \mid Q\{^{z}/_{w}\} \mid R). \tag{1}$$

Note that the rule dictates that the two involved processes should be contiguous. The presence of the process R allows any number of other processes to be there in parallel under the scope restriction while the communication happens. The *substitution* $\{^{z}/_{w}\}$ on process Q means that all free occurrences of w in Q are replaced by z. See also Examples 2 and 3.

The rule described in (1) has two versions in Figure 3. Rule [R-$LinCom$] models linear communication, and Rule [R-$UnCom$] models the input of a replicated process. Recall that the lin-predicate is often omitted. Note that the replicated process continues to exist after the reduction, but a fourth parallel process has also spawned to which the substitution is applied.

Example 2. Consider the process P in Example 1. Applying the rule [R-$UnCom$] in Figure 3 gives us the following reduction. Before we can apply the rule [R-$UnCom$], observe that there should be a third process running in parallel with the threads doing the output and input. For this, we apply the first axiom of Figure 2.

$$\begin{array}{c}(\boldsymbol{\nu} xy)(\overline{x}\langle \mathsf{true}\rangle \,.\, x(v) \,.\, Q' \mid \mathsf{un}\; y(v) \,.\, \overline{y}\langle v\rangle \,.\, \mathbf{0}) \\ \equiv \\ (\boldsymbol{\nu} xy)(\overline{x}\langle \mathsf{true}\rangle \,.\, x(v) \,.\, Q' \mid \mathsf{un}\; y(v) \,.\, \overline{y}\langle v\rangle \,.\, \mathbf{0} \mid \mathbf{0})\end{array}$$

On this last process, we can apply [R-$UnCom$] to reduce the process:

$$\begin{array}{rl}(\boldsymbol{\nu} xy)(\overline{x}\langle \mathsf{true}\rangle \,.\, x(v) \,.\, Q' \mid \mathsf{un}\; y(v) \,.\, \overline{y}\langle v\rangle \,.\, \mathbf{0} \mid \mathbf{0}) & \\ \rightarrow & [R\text{-}UnCom] \\ (\boldsymbol{\nu} xy)(x(v) \,.\, Q' \mid \overline{y}\langle \mathsf{true}\rangle \,.\, \mathbf{0} \mid \mathsf{un}\; y(v) \,.\, \overline{y}\langle v\rangle \,.\, \mathbf{0} \mid \mathbf{0}) &\end{array}$$

Notice that in the continuation process $\overline{y}\langle v\rangle \,.\, \mathbf{0}$ after the input on channel y, the value true has replaced the variable v. Notice also that the replicated process is still there in parallel. We can now apply the axiom for commutativity and neutral element before applying Rule [R-$LinCom$] and reduce the process once more.

$$\begin{array}{rl}(\boldsymbol{\nu} xy)(\overline{y}\langle \mathsf{true}\rangle \,.\, \mathbf{0} \mid x(v) \,.\, Q' \mid \mathsf{un}\; y(v) \,.\, \overline{y}\langle v\rangle \,.\, \mathbf{0}) & \\ \rightarrow & [R\text{-}LinCom] \\ (\boldsymbol{\nu} xy)(\mathbf{0} \mid Q'\{^{\mathsf{true}}/_{v}\} \mid \mathsf{un}\; y(v) \,.\, \overline{y}\langle v\rangle \,.\, \mathbf{0}) &\end{array}$$

The axiom for neutral element can be applied again to collect the terminated thread.

$$(\boldsymbol{\nu} xy)(\mathbf{0} \mid Q'\{^{\mathsf{true}}/_{v}\} \mid \mathsf{un}\; y(v) \,.\, \overline{y}\langle v\rangle \,.\, \mathbf{0}) \equiv (\boldsymbol{\nu} xy)(Q'\{^{\mathsf{true}}/_{v}\} \mid \mathsf{un}\; y(v) \,.\, \overline{y}\langle v\rangle \,.\, \mathbf{0})$$

Instead of applying axioms of structural congruence before and after reduction of processes each time, like in Example 2, Rule [R-$Struct$] is introduced. We also allow communications to happen under restrictions of other names, and in parallel composition with other processes, as reflected by Rules [R-Res] and [R-Par] respectively.

Example 3. One of the powers of the π-calculus is that it allows us to exchange communication endpoints, which is called *delegation*. Consider a server that can repeatedly receive a channel endpoint. Upon receiving a channel, the spawned copy of the server will send something over this channel and then terminate.

$$\mathsf{Server} \triangleq \mathsf{un}\ y(a)\ .\ \overline{a}\langle \mathsf{true}\rangle\ .\ \mathbf{0}$$

The channel y represents a publicly available channel. A client can then communicate with this server by instantiating a private session using the restriction operator, after which it passes one end of this session to the server. After this, it can receive the value from the server and continue as a process P'.

$$\mathsf{Client} \triangleq (\boldsymbol{\nu} sc)\overline{x}\langle s\rangle\ .\ c(v)\ .\ P'$$

The system that models both server and the client is then described by writing $(\boldsymbol{\nu} xy)(\mathsf{Server}\ \mid\ \mathsf{Client})$, and this process reduces as follows:

$$\begin{array}{c} (\boldsymbol{\nu} xy)(\mathsf{un}\ y(a)\ .\ \overline{a}\langle \mathsf{true}\rangle\ .\ \mathbf{0}\ \mid\ (\boldsymbol{\nu} sc)\overline{x}\langle s\rangle\ .\ c(v)\ .\ P') \\ \rightarrow \qquad\qquad [R\text{-}Struct] \\ (\boldsymbol{\nu} xy)(\mathsf{un}\ y(a)\ .\ \overline{a}\langle \mathsf{true}\rangle\ .\ \mathbf{0}\ \mid\ (\boldsymbol{\nu} sc)(\overline{s}\langle \mathsf{true}\rangle\ .\ \mathbf{0}\ \mid\ c(v)\ .\ P')). \end{array}$$

Note that we applied scope extrusion of $(\boldsymbol{\nu} sc)$ here. After this communication, a private session between (a copy of) the server and the client is established.

The rules that we introduced describe the *monadic* π-calculus, which means that only a single name can be communicated on each synchronization.

Notation 3. We let $\widetilde{u}, \widetilde{v}, \ldots$ range over *sequences* of names, and the length of a sequence of variables $\widetilde{u}$ is denoted by $|\widetilde{u}|$.

The syntax and semantics is naturally extended to support communication of a sequence of variables. This allows us to also model *polyadicity*: communication of multiple variables at once.

Figure 1 allows us to model processes in which two communicating channels are both trying to do an output, as in

$$(\boldsymbol{\nu} xy)(\overline{x}\langle \mathsf{true}\rangle\ .\ P\ \mid\ \overline{y}\langle \mathsf{false}\rangle\ .\ Q). \tag{2}$$

This process deadlocks, because there is no rule that allows the process to reduce. Another example of a bad process is

$$(\boldsymbol{\nu} xy)(\overline{x}\langle \mathsf{false}\rangle\ .\ P\ \mid\ \overline{x}\langle \mathsf{true}\rangle\ .\ Q\ \mid\ \mathsf{lin}\ y(b)\ .\ R), \tag{3}$$

in which two threads are trying to use the same linear channel to output a basic type value. The process can reduce in two different ways, which is an undesirable situation. The next section introduces a type system that will rule out this kind of bad processes.

3 Type System

A type system for the π-calculus consists of a set of rules that can be used to prove that all names in a process P are typed in such a way that all channels follow their protocol. This is an important step in formally verifying safe communication of programs.

3.1 Terminology

Types are assigned to names by *judgments*. If x is a name in a process P, the judgment $x : T$ intuitively denotes that x is of type T in P. Note that if x is a channel, the type T will change as P evolves. A *typing context* Γ, or simply *context*, is an unordered collection of judgments, and assigns types to *free* variables of a process P.

A process P can then be *typed* by a Γ. This means that the statement $\Gamma \vdash P$, pronounced "gamma types P", can be proven using the typing rules of the type system. A process P is then said to be *well-typed*, or *typable*, if there exists a Γ that proves that $\Gamma \vdash P$ with the rules of the type system.

Using the rules, a proof tree can be built that decomposes the syntax of a process, depending on its structure by one of the syntax rules in Figure 1. The typing rules can be found in Figure 8 of Section 3.5, and are explained throughout this entire section.

Naturally, the number of typable processes that we can model will be smaller than the number of untyped processes that we can model. For example, the processes in (2) and (3) will be ruled out by a proper type system.

3.2 Syntax

The syntax of types is described in Figure 4. We can divide types into two categories: session types for channels and basic types for basic type variables. In our case, bool is the only basic type, but the system can easily be extended to support any other basic type.

Session types can be subdivided into two categories, based on the qualifier they have. There exist linear pretypes and unrestricted pretypes. Linear pretypes are for channels that are used in one thread only, and unrestricted pretypes can be used by multiple threads. This distinction is used to be able to type processes with replication. For replication we also need recursive types, and these are explained in Section 3.6.

Notation 4. Similarly for sequences of variables, we will also use the notation for list of types, written $\widetilde{T} = (T_1, \ldots, T_n)$. Then $|\widetilde{T}|$ denotes the length of the list, and a series of judgments $x_1 : T_1, \ldots x_n : T_n$ will be abbreviated with $\widetilde{x} : \widetilde{T}$, where we always have that $|\widetilde{x}| = |\widetilde{T}|$.

A pretype describes the protocol for a channel. A question mark indicates that a channel is expected to receive a sequence with the list of types. For example, if $x : \mathsf{lin}\ ?\widetilde{T}\ .\ \mathsf{end}$, then x is a linear channel that inputs some sequence $\widetilde{u}$ such that $\widetilde{u} : \widetilde{T}$. After this input is done, the session will end,

$T ::=$		**Types:**
	$q\ p$	session type
	end	termination
	a	type variable
	$\mu a.T$	recursive type
	bool	basic type
$q ::=$		**Qualifiers:**
	lin	linear
	un	unrestricted
$p ::=$		**Pretypes:**
	$?(T, \ldots, T).T$	receive
	$!(T, \ldots, T).T$	send
$\Gamma ::=$		**Contexts:**
	$\emptyset$	empty context
	$\Gamma, x : T$	assumption

Figure 4: Syntax of types

which means that x cannot occur anywhere else in the process. The exclamation mark says that a channel is supposed to *send* a sequence of names with the types in the list.

Contexts contain all judgments that can prove that a process is typable, and writing $\Gamma_1 = \Gamma_2, x : T$ can be seen as doing the *assumption* that Γ_1 contains the judgment $x : T$. We let $\Gamma, \Delta, \ldots$ range over contexts.

Notation 5. For a context Γ, $\mathrm{dom}(\Gamma)$ denotes the set of variables x such that $x : T$ is in Γ, for some T.

The following example provides an intuition on how the syntax of types and contexts is used.

Example 4. Let Q be the process from Example 1, so $Q = \overline{x}\langle \mathsf{true} \rangle\ .\ x(v)\ .\ Q'$. Assume that Q' is typable. Then there exists a Γ such that $\Gamma \vdash Q'$. If $v \in \mathtt{fn}(Q')$ then $v : T \in \Gamma$ for some type T. We can thus write

$$\Gamma = \Delta, v : T,$$

for some context Δ. On the other hand, we assume $x \notin \mathtt{fn}(Q')$ and hence $x \notin \mathrm{dom}(\Gamma)$. The following statement intuitively shows that $x(v)\ .\ Q'$ is typable by providing a context that types it:

$$\Delta, x : \mathsf{lin}\ ?T\ .\ \mathsf{end} \vdash x(v)\ .\ Q'$$

In words, the process $x(v)\ .\ Q'$ is typed by a context saying that x acts as a linear channel that receives something of type T and then stops. We have used Δ instead of Γ with a judgment for x to type the process. This is because $v : T \in \Gamma$, and contexts only contain judgments for free names, but v is bound in $x(v)\ .\ Q'$. The reader should now be convinced that if Q' is typable, then the process Q from Example 1 is typable:

$$\Delta, x : \mathsf{lin}\ !\mathsf{bool}\ .\ \mathsf{lin}\ ?T\ .\ \mathsf{end} \vdash \overline{x}\langle \mathsf{true} \rangle\ .\ x(v)\ .\ Q'$$

$$\overline{q?\widetilde{T}.U} = q!\widetilde{T}.\overline{U} \qquad \overline{q!\widetilde{T}.U} = q?\widetilde{T}.\overline{U} \qquad \overline{\mathsf{end}} = \mathsf{end}$$

$$\overline{\mu a.T} = \mu a.\overline{T} \qquad \overline{a} = a$$

Figure 5: Dual function on types

Channel x will receive something of type T, because the judgment $v : T$ is in Γ. Note that typability is still an intuitive notion at this point. We cannot yet formally prove the statement, because we have yet to introduce the rules for this.

3.3 Type Duality

A restriction operator creates two new names as channel endpoints that communicate with each other. For well-typed processes, this means that every type that one endpoint inputs, the other endpoint should output, and vice versa. This can be made formal using *duality* of types, which is inductively defined in Figure 5.

Dual types share the same qualifier, lin or un, and if one channel outputs a type T then the other inputs T, and vice versa. The continuation of the session types are recursively defined to be each others dual as well. Duality for recursive types are explained later. Consider the following example:

Example 5. Consider the process P from Example 1 and assume that $Q' = \mathbf{0}$ and R is not replicated:

$$P = (\boldsymbol{\nu} xy)(\overline{x}\langle \mathsf{true} \rangle \,.\, x(v) \,.\, \mathbf{0} \mid y(v) \,.\, \overline{y}\langle v \rangle \,.\, \mathbf{0})$$

This simple process models two threads passing a variable back and forth. The only free name that occurs in P is the constant basic type value true. Hence, it is easily conjectured that P can be proven to be well-typed by starting with the following statement:

$$\mathsf{true} : \mathsf{bool} \vdash (\boldsymbol{\nu} xy)(\overline{x}\langle \mathsf{true} \rangle \,.\, x(v) \,.\, \mathbf{0} \mid y(v) \,.\, \overline{y}\langle v \rangle \,.\, \mathbf{0})$$

Rule $[T\text{-}Res]$ from Figure 8 dictates that this process is well-typed if we can prove the following assumption:

$$\mathsf{true} : \mathsf{bool}, x : T, y : \overline{T} \vdash \overline{x}\langle \mathsf{true} \rangle \,.\, x(v) \,.\, \mathbf{0} \mid y(v) \,.\, \overline{y}\langle v \rangle \,.\, \mathbf{0},$$

where the restriction has been removed from the process, and the context contains two judgments saying that x and y are dual endpoints. By applying rules subsequently, we build a proof tree that proves that a context types a process. The proof tree is complete when no more rules can be applied. In the following tree, which is incomplete, the statement below the line is called the *conclusion*, and the statement above the line is called an *assumption*.

$$\frac{\mathsf{true} : \mathsf{bool}, x : T, y : \overline{T} \vdash \overline{x}\langle \mathsf{true} \rangle \,.\, x(v) \,.\, \mathbf{0} \mid y(v) \,.\, \overline{y}\langle v \rangle \,.\, \mathbf{0}}{\mathsf{true} : \mathsf{bool} \vdash (\boldsymbol{\nu} xy)(\overline{x}\langle \mathsf{true} \rangle \,.\, x(v) \,.\, \mathbf{0} \mid y(v) \,.\, \overline{y}\langle v \rangle \,.\, \mathbf{0})}\ [T\text{-}Res]$$

$$\frac{}{\emptyset = \emptyset \circ \emptyset}\ [S\text{-}Empty] \qquad \frac{\Gamma = \Gamma_1 \circ \Gamma_2 \qquad \mathsf{un}(T)}{\Gamma, x : T = (\Gamma_1, x : T) \circ (\Gamma_2, x : T)}\ [S\text{-}Un]$$

$$\frac{\Gamma = \Gamma_1 \circ \Gamma_2}{\Gamma, x : \mathsf{lin}\ p = (\Gamma_1, x : \mathsf{lin}\ p) \circ \Gamma_2}\ [S\text{-}LinA] \qquad \frac{\Gamma = \Gamma_1 \circ \Gamma_2}{\Gamma, x : \mathsf{lin}\ p = \Gamma_1 \circ (\Gamma_2, x : \mathsf{lin}\ p)}\ [S\text{-}LinB]$$

Figure 6: Rules defining context split, $\Gamma = \Gamma \circ \Gamma$

To finish the proof, we must find a suitable type T and apply more rules until the tree is complete. A sensible choice for T would be lin !bool . lin ?bool . end, for which the reader can verify that $\overline{T} =$ lin ?bool . lin !bool . end by the rules in Figure 5.

Duality is not defined for basic types. This is to rule out nonsensical types such as lin !bool . bool in well-typed processes.

The only rule that can be applied now to the assumption in Example 5 is one for parallel composition. Because linear channels can be used in one thread only, the typing rule for parallel will split the context into two parts. Judgments for linear channels will be used to type one process only, and unrestricted channels will be used in both. Before introducing formal rules for this, we introduce the following predicate that checks whether a type is unrestricted:

$$\mathsf{un}(T) \text{ if and only if } T = \mathsf{bool} \text{ or } T = \mathsf{end} \text{ or } T = \mathsf{un}\ p.$$

Note that all basic types are unrestricted. A context Γ is unrestricted, written $\mathsf{un}(\Gamma)$, if the type of every judgment in it is unrestricted: $x : T \in \mathsf{un}(\Gamma)$ implies $\mathsf{un}(T)$. For a type T, the predicate $\mathsf{lin}(T)$ is defined to be always true, and similarly for contexts.

3.4 Context Operators

Context *split* is a binary operator on contexts. It is defined by the set of rules described in Figure 6. The first rule says that an empty context cannot be split further. The second rule has two assumptions: Γ can be split into two contexts Γ_1 and Γ_2, and T is an unrestricted type. Under these two assumptions, the context $\Gamma, x : T$ can be split in the same way, and $x : T$ goes into both contexts. The other two rules make sure that judgments with linear session types go into exactly one of the two contexts. Consider the following example:

Example 6. In order to apply Rule $[T\text{-}Par]$ from Figure 8 on the assumption in Example 5, we try to find Γ_1 and Γ_2 such that $\Gamma_1 \circ \Gamma_2 = \Gamma$, where

$$\Gamma = x : T, y : \overline{T}, \mathsf{true} : \mathsf{bool}$$

and $T =$ lin !bool . lin ?bool . end. Intuitively, we know that judgments for a linear type go into exactly one context, whereas unrestricted type judgments go into both. Since T is

linear, the following splitting is easily conjectured:

$$\Gamma_1 = x : T, \mathsf{true} : \mathsf{bool} \qquad \Gamma_2 = y : \overline{T}, \mathsf{true} : \mathsf{bool}$$

To show that this splitting is correct, we build a tree to prove it using the rules from Figure 6 as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\emptyset = \emptyset \circ \emptyset}\,[S\text{-}Empty]}{x : T = (x : T) \circ \emptyset}\,[S\text{-}LinA]}{x : T, y : \overline{T} = (x : T) \circ (y : \overline{T})}\,[S\text{-}LinB] \qquad \dfrac{}{\mathsf{un}(\mathsf{bool})}}{x : T, y : \overline{T}, \mathsf{true} : \mathsf{bool} = (x : T, \mathsf{true} : \mathsf{bool}) \circ (y : \overline{T}, \mathsf{true} : \mathsf{bool})}\,[S\text{-}Un]$$

This proves that $\Gamma = \Gamma_1 \circ \Gamma_2$. To continue the proof that the process P from Example 5 is typable, we apply Rule $[T\text{-}Par]$ from Figure 8 as follows:

$$\dfrac{\mathsf{true} : \mathsf{bool}, x : T \vdash \overline{x}\langle \mathsf{true} \rangle \,.\, x(v) \,.\, \mathbf{0} \qquad \mathsf{true} : \mathsf{bool}, y : \overline{T} \vdash y(v) \,.\, \overline{y}\langle v \rangle \,.\, \mathbf{0}}{\mathsf{true} : \mathsf{bool}, x : T, y : \overline{T} \vdash \overline{x}\langle \mathsf{true} \rangle \,.\, x(v) \,.\, \mathbf{0} \mid y(v) \,.\, \overline{y}\langle v \rangle \,.\, \mathbf{0}}\,[T\text{-}Par]$$

The proof should continue with both assumptions on either side of the tree.

The context split operator can be shown to be commutative as well as associative using straightforward induction [12].

Notation 6. For a context Γ, we will use the following notations:

- $\mathcal{U}(\Gamma) \subset \Gamma$ denotes the collection of judgments $x : T \in \Gamma$ such that $\mathsf{un}(T)$ holds.
- $\mathcal{L}(\Gamma) \subset \Gamma$ denotes the collection of judgments $x : T \in \Gamma$ such that $T = \mathsf{lin}\ p$, for some p.

With this notation, we prove the following lemma that describes two important properties of the context split operator:

Lemma 1 (*Properties of context split*)**.** If $\Gamma = \Gamma_1 \circ \Gamma_2$ then

1. $\mathcal{U}(\Gamma) = \mathcal{U}(\Gamma_1) = \mathcal{U}(\Gamma_2)$, and
2. If $x : \mathsf{lin}\ p \in \Gamma$ then either $x : \mathsf{lin}\ p \in \Gamma_1$ and $x \notin \mathrm{dom}(\Gamma_2)$, or $x : \mathsf{lin}\ p \in \Gamma_2$ and $x \notin \mathrm{dom}(\Gamma_1)$.

Proof. The proof is by induction on the structure of the context Γ. The base case is $\Gamma = \emptyset$ and properties (1) and (2) are satisfied immediately by $[S\text{-}Empty]$. For the inductive step, we write $\Gamma = \Delta, x : T$. By the induction hypothesis, we may assume that both properties hold for $\Delta = \Delta_1 \circ \Delta_2$. The proof of the inductive step for property (1) is as follows: $x : T \in \mathcal{U}(\Gamma)$ if and only if $\mathsf{un}(T)$, and similarly for Γ_1 and Γ_2 by applying Rule $[S\text{-}Un]$. Considering property (2), if $T = \mathsf{lin}\ p$ for some p, then $\mathsf{un}(T)$ does not hold, and the only two rules that we can apply is either $[S\text{-}LinA]$ or $[S\text{-}LinB]$. This concludes the proof. ☺

From this we can derive the property that if Γ splits into Γ_1 and Γ_2, then $\Gamma = \Gamma_1, \mathcal{L}(\Gamma_2)$. A comma can be used interchangeably with notation for union:

$$\frac{x : U \notin \Gamma}{\Gamma + x : T = \Gamma, x : T} \qquad \frac{\mathsf{un}(T)}{(\Gamma, x : T) + x : T = \Gamma, x : T}$$

Figure 7: Rules defining context update, $\Gamma = \Gamma + x : T$

Notation 7. The union of judgments $\Gamma = \bigcup_{i \in I} x_i : T_i$ denotes a context.

If x is the subject of communication in a prefix α, the type of x in a process $\alpha \,.\, P$ is different from the type of x in P. To prove typability for the case of output and input, we therefore need another context operator called the context *update*, that describes the transformation of a type.

The context update works on a context with a judgment, and makes distinction between linear and unrestricted types. The rules can be found in Figure 7. Note that if T is unrestricted, then the operator is only defined if the type that is already in the context is equal to the type to which it is updated. The following example illustrates an important implication of this.

Example 7. Consider the process P, with the replicated process R and two variants of Q from Example 1:

$$R = \mathsf{un}\ y(v) \,.\, \overline{y}\langle v \rangle \,.\, \mathsf{end}$$
$$Q_{\mathsf{true}} = \overline{x}\langle \mathsf{true} \rangle \,.\, x(b) \,.\, Q_1$$
$$Q_{\mathsf{false}} = \overline{x}\langle \mathsf{false} \rangle \,.\, x(b) \,.\, Q_2$$
$$P = (\boldsymbol{\nu} xy)(Q_{\mathsf{true}} \mid Q_{\mathsf{false}} \mid R)$$

This process can reduce in two steps as follows, where both Q_{true} and Q_{false} will do an output, and R will spawn twice a copy of its continuation:

$$P \to\to (\boldsymbol{\nu} xy)(x(b) \,.\, Q_1 \mid x(b) \,.\, Q_2 \mid \overline{y}\langle \mathsf{true} \rangle \,.\, \mathsf{end} \mid \overline{y}\langle \mathsf{false} \rangle \,.\, \mathsf{end} \mid R)$$

Taking a moment to reflect on this, who will receive true and who will receive false, Q_1 or Q_2, is non-deterministic. The type system also rules out this undesirable situation. To illustrate this, we will show that R is *not* typable. Assume for contradiction that R is typable, then there exists Γ such that $\Gamma \vdash R$. Since R starts with an input, the only rule from Figure 8 we can apply to $\Gamma \vdash R$ is $[T\text{-}In]$. This is done in the following way.

$$\dfrac{\mathsf{un}(\Gamma) \qquad \Gamma_1 \vdash y : q?T \,.\, U \qquad \dfrac{(4)}{(\Gamma_2 + y : U), v : T \vdash \overline{y}\langle v \rangle \,.\, \mathsf{end}}\,[T\text{-}Out]}{\Gamma \vdash \mathsf{un}\ y(v) \,.\, \overline{y}\langle v \rangle \,.\, \mathsf{end}}\,[T\text{-}In]$$

where $\Gamma = \Gamma_1 \circ \Gamma_2$. Since we know that Γ is unrestricted, we have $\mathcal{U}(\Gamma) = \Gamma$. Then, using Lemma 1, we have $\Gamma_1 = \Gamma_2 = \Gamma$. Then $y : \mathsf{un}\ ?T \,.\, U \in \Gamma_2$, and Rule $[T\text{-}In]$ implies that $U = \mathsf{un}\ ?T \,.\, U$, and so $\Gamma_2 + y : U = \Gamma_2, y : \mathsf{un}\ ?T \,.\, U$. The only rule we can apply on the right side is $[T\text{-}Out]$, so we continue building the proof tree in the only possible way.

$$\Gamma_3 \vdash y : !T \,.\, U \qquad \Gamma_4 \vdash v : T \qquad \Gamma_5 + y : U \vdash \mathbf{0} \tag{4}$$

Here, $\Gamma_3 \circ \Gamma_4 \circ \Gamma_5 = \Gamma_2, y : \mathsf{un}\ ?T \,.\, U$. Also, because Γ_2 is unrestricted, we have $\Gamma_3 = \Gamma_2, y : \mathsf{un}\ ?T \,.\, U$ as before. But then $y : \mathsf{un}\ ?T.U \in \Gamma_3$ which contradicts the

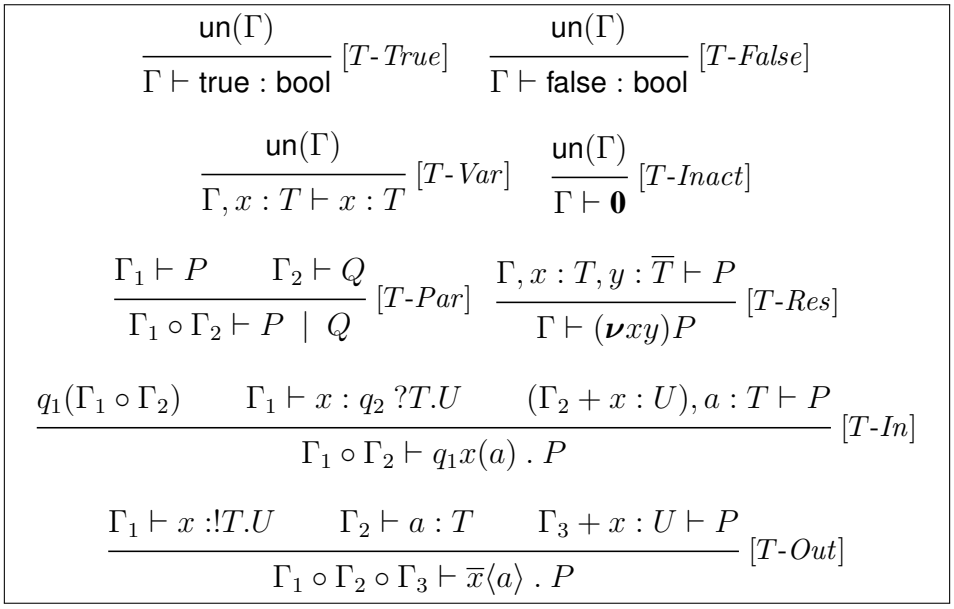

Figure 8: Typing rules

assumption in the proof tree $\Gamma_3 \vdash y :!T.U$. We have proceeded through the only possible way, applying Rule $[T\text{-}In]$ and Rule $[T\text{-}Out]$ respectively, and it led to a contradiction. The original assumption that R is typed by an existing Γ is therefore impossible in the type system. Hence, R is not typable.

3.5 Typing Rules

The syntax of types has been presented, and the context operators split and update have been motivated and explained. This makes us ready to present the typing rules, which can be found in Figure 8.

All rules except the base cases for variables and inaction have been illustrated in the preceding examples. The base cases ensure that all linear sessions have been "used up", i.e., there is no linear session that still has to perform an action. For example, we have $x : \mathsf{end} \vdash \mathbf{0}$, but $x : \mathsf{lin}\ ?T.\mathsf{end} \not\vdash \mathbf{0}$, because the type is not unrestricted, and x has not finished its session.

Note that some assumptions are judgments on processes, whereas others are judgments on variables. For judgments on variables, the only rules that can be applied are $[T\text{-}True]$, $[T\text{-}False]$, and $[T\text{-}Var]$. On the other hand, judgments for processes always expand the proof tree, except when $P = \mathbf{0}$.

Notation 8. Without causing ambiguity, the lin qualifier and the trailing ".end" for session types are omitted. The un qualifier is always written explicitly.

Notation 9. Linear communication of nothing is typed by !end and ?end.

For example: $x :!\mathsf{end}, y :?\mathsf{end} \vdash \overline{x}\langle\rangle\ .\ \mathbf{0} \mid y()\ .\ \mathbf{0}$

Example 8. We continue the proof tree for P from Examples 5 and 6 to show that it is typable. We first show that $\Gamma_1 \vdash Q$ with the given Γ_1 in Example 6, and continue the proof tree on the left-hand side.

$$\dfrac{\dfrac{\mathsf{un}(\mathsf{true} : \mathsf{bool})}{\mathsf{true} : \mathsf{bool}, x : T \vdash x : T}\,[T\text{-}Var] \quad \dfrac{\mathsf{un}(\mathsf{true} : \mathsf{bool})}{\mathsf{true} : \mathsf{bool} \vdash \mathsf{true} : \mathsf{bool}}\,[T\text{-}True] \quad (5)}{\mathsf{true} : \mathsf{bool}, x : !\mathsf{bool}.?\mathsf{bool} \vdash \overline{x}\langle \mathsf{true} \rangle \,.\, x(v) \,.\, \mathbf{0}}\,[T\text{-}Out]$$

where $T = !\mathsf{bool}.?\mathsf{bool}$. The tree (5) is given as follows:

$$\dfrac{\mathsf{lin}(\Gamma') \quad \dfrac{\mathsf{un}(\mathsf{true} : \mathsf{bool})}{\mathsf{true} : \mathsf{bool}, x : ?\mathsf{bool} \vdash x : ?\mathsf{bool}}\,[T\text{-}Var] \quad \dfrac{\mathsf{un}(\mathsf{true} : \mathsf{bool}, x : \mathsf{end}, v : \mathsf{bool})}{\mathsf{true} : \mathsf{bool}, x : \mathsf{end}, v : \mathsf{bool} \vdash \mathbf{0}}\,[T\text{-}Inact]}{\mathsf{true} : \mathsf{bool}, x : ?\mathsf{bool} \vdash x(v) \,.\, \mathbf{0}}\,[T\text{-}In] \qquad (5)$$

where $\Gamma' = \mathsf{true} : \mathsf{bool}, x : ?\mathsf{bool}$. The lin predicate is always satisfied, and all un predicates are also satisfied. The reader can verify that all context splittings are correct, and the context update was also well-defined, since x is a linear channel. The process Q is thus proven to be typable by Γ_1. One similarly proves that $\Gamma_2 \vdash R$ (where R is not replicated) by applying first Rule $[T\text{-}In]$ and then Rule $[T\text{-}Out]$. Hence, P from Example 5 is typable.

Intuitively, the judgment $\mathsf{true} : \mathsf{bool}$ in the tree (5) in Example 8 could have been omitted, since it did not occur in $x(v) \,.\, \mathbf{0}$. This motivates the following lemma:

Lemma 2 (*Unrestricted weakening*)**.** If $\Gamma_1 \vdash P$ and $\mathsf{un}(\Gamma_2)$ then $\Gamma_1, \Gamma_2 \vdash P$.

Proof. This is a generalization for contexts of the result for single judgments used in [12]. ☺

It is called weakening because it adds seemingly useless information to a claim. It allows us to remove any judgments $x : T$ from the context when going up in a proof tree, as long as $\mathsf{un}(T)$.

Example 9. Considering the rightmost subtree in (5) from Example 8, we might also have provided the following proof tree:

$$\dfrac{\dfrac{\mathsf{un}(\emptyset)}{\emptyset \vdash \mathbf{0}}\,[T\text{-}Inact]}{\mathsf{true} : \mathsf{bool}, x : \mathsf{end}, v : \mathsf{bool} \vdash \mathbf{0}}\,\textsc{Lemma 2}$$

3.6 Recursive Types

The syntax rules for types in Figure 4 also describes recursive types, written $\mu a.T$. Recursive types describe an infinite structure, and are necessary to type unrestricted channels. Because of the infinite structure, equality of recursive types is defined co-inductively, and omitted here.

Like the symbol "ν" in processes, "μ" is a binder that introduces a new *type variable* for the type T. The binder gives rise to alpha-equivalence in the standard way, meaning for example that

$\mu a.\mathsf{un}\ !\mathsf{bool}.a = \mu b.\mathsf{un}\ !\mathsf{bool}.b$.

An infinite type can be unfolded using substitution: $\mu a.T = T\{\mu a.T\ /a\}$. For example, if $T = \mathsf{un}\ !\mathsf{bool}.a$ then $\mu a.T = \mathsf{un}\ !\mathsf{bool}.\mu a.T = \mathsf{un}\ !\mathsf{bool}.\mathsf{un}\ !\mathsf{bool}.\mu a.T$, etc. Consider also the following example:

Example 10. In this example, we show that the following two types are equivalent.

$$\begin{array}{rl} T_1 = & \mu a\ .\ \mathsf{un}\ !\mathsf{bool}\ .\ \mathsf{un}\ ?\mathsf{bool}\ .\ a \\ T_2 = & \mathsf{un}\ !\mathsf{bool}\ .\ \mu b\ .\ \mathsf{un}\ ?\mathsf{bool}\ .\ \mathsf{un}\ !\mathsf{bool}\ .\ b \end{array}$$

For this, consider their first unfolding by substitution:

$$\begin{array}{rll} T_1 = & \mathsf{un}\ !\mathsf{bool}\ .\ \mathsf{un}\ ?\mathsf{bool}\ . & \mu a\ .\ \mathsf{un}\ !\mathsf{bool}\ .\ \mathsf{un}\ ?\mathsf{bool}\ .\ a \\ T_2 = & \mathsf{un}\ !\mathsf{bool}\ .\ \mathsf{un}\ ?\mathsf{bool}\ . & \mathsf{un}\ !\mathsf{bool}\ .\ \mu b\ .\ \mathsf{un}\ ?\mathsf{bool}\ .\ \mathsf{un}\ !\mathsf{bool}\ .\ b \end{array}$$

We observe that $T_1 = \mathsf{un}\ !\mathsf{bool}.\mathsf{un}\ ?\mathsf{bool}.T_1$, and $T_2 = \mathsf{un}\ !\mathsf{bool}.\mathsf{un}\ ?\mathsf{bool}.T_2$. We conclude that the types have the same infinite unfolding, and that they are equivalent.

Example 7 showed that types such as $\mathsf{un}\ ?\mathsf{bool}.\mathsf{un}\ !\mathsf{bool}.\mu a.T$ are not supported in this system, due to Rule $[T\text{-}In]$. In fact, the only tail recursive type allowed in this system is of the form $\mu a.\alpha.a$, where $\alpha = \mathsf{un}\ !\widetilde{T}$ or $\alpha = \mathsf{un}\ ?\widetilde{T}$. Because of this, the following notation is introduced:

Notation 10. The type $*?\widetilde{T}$ denotes the recursive type $\mu a.\mathsf{un}?\widetilde{T}.a$, and similarly for $*!\widetilde{T}$.

4 Preliminaries

In Section 2 we have presented the π-calculus as a way to model communicating processes. After that, we introduced the type system by Vasconcelos as a step towards safe communication. In Section 5, we will introduce a function on processes that will induce minimal linear session types. For this, we need the definitions and notations presented in this section.

4.1 Notation

For well-typed processes, we know that every session and input can be assigned a type. This gives rise to the following convenient notation.

Notation 11. We let $(\boldsymbol{\nu} xy : T)$ denote the creation of two fresh names x of type T and y of type $\overline{T}$. We furthermore denote an input of a name z of type S over a channel x by $x(z : S)$.

Sequences of names are sometimes treated as totally ordered sets, and some notation for this should be justified.

Notation 12. Assume that the order of the names in $\widetilde{u}$ is the same as for the names in $\widetilde{v}$.

- Writing $\widetilde{u} \subset \widetilde{v}$ means that $x \in \widetilde{u}$ implies $x \in \widetilde{v}$.

- Also, $\widetilde{u} \setminus \widetilde{v}$ denotes the sequence of all names x such that $x \in \widetilde{u}$, but $x \notin \widetilde{v}$.

4.2 Definitions

A particular kind of typing context will be useful for the results.

Definition 1 (*Value environment*)**.** Let Γ be a context. We call Γ a *value environment* if $x : T \in \Gamma$ implies $T = \mathsf{bool}$.

This means that if $\Gamma \vdash P$ and Γ is a value environment, then all communication in P is done by sessions that are created in P. The communication is closed in some sense, and we refer to P as a *closed* process, or a *program*.

We need to decompose every channel x into a sequence of channels $\widetilde{x}$. To do this, it is useful to know the length that this sequence should have. We define the following function for types.

Definition 2 (*Session length function*)**.** For a type T, the function $\mathrm{ord}(T)$ is inductively defined as:

$$\begin{aligned} \mathrm{ord}(\mathsf{end}) &= 0 \\ \mathrm{ord}(a) &= 0 \\ \mathrm{ord}(\mathsf{bool}) &= 1 \\ \mathrm{ord}(\mu a \,.\, T) &= \mathrm{ord}(T) \\ \mathrm{ord}(q\ ?\widetilde{T}.U) &= 1 + \mathrm{ord}(U) \\ \mathrm{ord}(q\ !\widetilde{T}.U) &= 1 + \mathrm{ord}(U) \end{aligned}$$

For example, $\mathrm{ord}(!(\mathsf{bool}, \mathsf{bool}).!(?\mathsf{bool}.!\mathsf{bool}.\mathsf{end}).\mathsf{end}) = 2$. For recursive types, the function only works for types with *tail recursion*. Therefore, by Rule $[T\text{-}In]$, the function most likely always returns one.

Recall that our aim was to construct processes that only contain names with types that are minimal in some sense. Formally, this could be that we demand that for every type that occurs in P, we have $\mathrm{ord}(T) \leq 1$. This definition does not suffice, however, because delegated channels are going to be broken down as well. This asks for a recursive definition of a minimal type.

Definition 3 (*Minimal type*)**.** Let T be a type as in Figure 4. We say that T is a *minimal type* if $T = \mathsf{end}$, $T = \mathsf{bool}$, $T = a$, or

(i) If $T = q\ p$, then $p = !\widetilde{U}.\mathsf{end}$ or $p = ?\widetilde{U}.\mathsf{end}$, where all $U_i \in \widetilde{U}$ are minimal types,

(ii) If $T = \mu a.T'$, then $T' = q\ p$, where $p = !\widetilde{U}.a$ or $p = ?\widetilde{U}.a$, where all $U_i \in \widetilde{U}$ are minimal types.

We let M range over minimal types.

An example of a minimal session type is $\mu b \,.\, \mathsf{un}\ ?(\mathsf{bool}, !\mathsf{bool}\ .\ \mathsf{end})\ .\ b$. A nonexample is $\mathsf{lin}\ !(!\mathsf{bool}\ .\ ?\mathsf{bool}\ .\ \mathsf{end})\ .\ \mathsf{end}$.

Next, we introduce a function that decomposes types into a list of minimal types.

Definition 4 (*Type decomposition function*)**.** Let $\mathscr{T}$ be the set of types generated by Figure 4, and $\mathcal{P}(\mathscr{T})$ denote the set of all possible finite lists of types. The type decomposition function $\mathcal{G} : \mathscr{T} \to \mathcal{P}(\mathscr{T})$ is defined in the following way:

$$\begin{aligned}
\mathcal{G}(\mathsf{bool}) &= \mathsf{bool}, \\
\mathcal{G}(\mathsf{end}) &= \mathsf{end}, \\
\mathcal{G}(q\,!T\,.\,S) &= \begin{cases} q\,!\widetilde{U}\,.\,\mathsf{end}, & \text{if } S = \mathsf{end} \\ q\,!\widetilde{U}\,.\,\mathsf{end}, \mathcal{G}(S), & \text{otherwise,} \end{cases} \\
\mathcal{G}(q\,?T\,.\,S) &= \begin{cases} q\,?\widetilde{U}\,.\,\mathsf{end}, & \text{if } S = \mathsf{end} \\ q\,?\widetilde{U}\,.\,\mathsf{end}, \mathcal{G}(S), & \text{otherwise,} \end{cases}
\end{aligned}$$

where $\widetilde{U} = \mathcal{G}(T)$.

Consider the following example.

$$\mathcal{G}(!\mathsf{bool}\,.\,!(?\mathsf{bool}\,.\,!\mathsf{bool}\,.\,\mathsf{end})\,.\,\mathsf{end}) = !\mathsf{bool}\,.\,\mathsf{end}, !(?\mathsf{bool}\,.\,\mathsf{end}, !\mathsf{bool}\,.\,\mathsf{end})\,.\,\mathsf{end}$$

which indeed returns a list of minimal type.

5 Trios

A *trio* is a term that consists of exactly three sequential prefixes. This means we can always write a trio in the form $\alpha\,.\,\beta\,.\,\gamma\,.\,\mathbf{0}$. In this section we present a function that creates a trio for each term in a replication-free process P. The trio breakdown function maps the syntax of a monadic process to a polyadic process. The process P is assumed to be well-typed by a context Γ, where Γ contains only basic type judgments. In other words, P has no free names that perform communication.

Every trio that we introduce is triggered by an *activator* channel that we label n_k for some index k. After the communication has been performed, the *forward* channel m_{k+1} triggers the next trio, whose activator channel will be n_{k+1}. Just as in [7], a sequence $\widetilde{u}$ is used to pass names from one trio to the next, accumulating more and more names that are bound in P, after applying the rules for input and restriction. For a terminated process, the function creates a trio that will immediately terminate after the activator channel has synchronized. This rule and others can be found formally in Figure 9.

5.1 Output and Input

When an output prefix $\overline{x_i}\langle y\rangle$ is encountered, and $y : T \in \Gamma$, we create a trio that outputs on the same channel x_i. If $T = \mathsf{bool}$, nothing interesting happens, and the output will be simply y. If $T \neq \mathsf{bool}$, then y should be decomposed into a sequence $\widetilde{y}$, where the length is inferred through its type, $|\widetilde{y}| = \mathrm{ord}(T)$. Note that it might be that $y \in \widetilde{u}$, in which case we will use $\widetilde{y}$ such that $\widetilde{y} \subset \widetilde{u}$, and y is the first element of $\widetilde{y}$. In the continuation Q, we do not need x_i anymore, so it

is substituted by x_{i+1} in Q. This is the key difference with Parrow's definition of the breakdown function [7].

For an input $x_i(y : T)$, the new trio does an input on x_i and the same substitution $\{x_{i+1}\ /x_i\}$ is done on the breakdown of the continuation Q. The bound name y in the original process is replaced by a sequence $\widetilde{y}$ in the trio, where again $|\widetilde{y}| = \text{ord}(T)$. The sequence $\widetilde{u}$ will be concatenated with the input sequence $\widetilde{y}$ in the subsequent breakdown.

For both input and output, the channel name that is used will not be used in the continuation. Consequently, it can be removed from the sequence $\widetilde{u}$. This is done by writing $\widetilde{u} = \widetilde{u_1} x_i \widetilde{u_2}$ for some $\widetilde{u_1}, \widetilde{u_2}$, and continuing the breakdown of the process with just the sequence $\widetilde{u_1}\widetilde{u_2}$.

5.2 Parallel

If $P = Q \mid R$, then the function makes a trio containing one activator channel n_k, and two forward channels m_{k+1} and m_l. Both of these will activate the subsequent breakdown of one parallel component. The index l is chosen such that $l - k = |Q| + 1$. The sequences $\widetilde{u_Q}$ and $\widetilde{u_R}$ are used for the breakdowns of Q and R respectively. They are constructed as follows. If $\Gamma_Q \circ \Gamma_R \vdash Q \mid R$, and $\Gamma_Q \vdash Q$ and $\Gamma_R \vdash R$, then $\widetilde{u_Q} := \widetilde{u} \setminus \text{dom}(\mathcal{L}(\Gamma_R))$, and $\widetilde{u_R} := \widetilde{u} \setminus \text{dom}(\mathcal{L}(\Gamma_Q))$. In words, $\widetilde{u_Q}$ contains all unrestricted names of $\widetilde{u}$ and all linear names that are used in Q, and $\widetilde{u_R}$ contains all linear names that are used in R and all unrestricted names.

5.3 Restriction

Parrow worked in an untyped π-calculus, where the rule for restriction created a trio that would invoke the name inventor. In our case, the name inventor will consist of several name servers composed in parallel, one server for each session type in the context that types P. Our trio breakdown function then works on a subterm starting with a restriction operator in a way that makes use of one of the name servers, depending on the type.

The idea of a typed name server is that it invents a series of names with fixed types that it sends over a channel received by a trio. The trio should thus create a private session and send one channel end to the name server. After receiving the new names, the names are appended to the sequence $\widetilde{u}$ that is passed along to be used by other trios.

The channel in a name server for a type T is named s_T. It is supposed to synchronize with the client channel c_T in a trio that needs new names.

Definition 5 (*Name server*)**.** The name server for a type T is defined as

$$N_T \triangleq \mathsf{un}\ s_T(z : U_T)\ .\ (\boldsymbol{\nu} x_1 y_1 : V_{T,1}) \ldots (\boldsymbol{\nu} x_t y_t : V_{T,t}) \overline{z}\langle \widetilde{x}\widetilde{y} \rangle\ .\ \mathbf{0},$$

where $\widetilde{V_T} = \mathcal{G}(T)$.

The channel z in each N_T is then typed by $U_T = \mathsf{lin}\ !(\widetilde{V_T}\widetilde{\overline{V_T}}).\mathsf{end}$, where we let $\widetilde{V_T}$ and $\widetilde{\overline{V_T}}$ denote the sequences $V_{T,1} \ldots V_{T,t}$ and $\overline{V_{T,1}} \ldots \overline{V_{T,t}}$ respectively, with $t = |\widetilde{V_T}|$.

Definition 6 (*Name inventor*)**.** Let the types $T_1, \ldots, T_q$ be given such that $(\boldsymbol{\nu} xy : T_i)$ for $i \in \{1, \ldots, q\}$ occurs at least once in P. Then the name inventor for P is given by

$$N_P \triangleq N_{T_1} \mid \cdots \mid N_{T_q},$$

where each name server N_{T_i} is a name server as in Definition 5.

In order for a trio to have access to one of the name servers in the name inventor, covariable restrictions for each type T_i are added to the translation in Section 6. This is done by a series of restrictions $(\boldsymbol{\nu} s_{T_1} c_{T_1} : S_1) \ldots (\boldsymbol{\nu} s_{T_q} c_{T_q} : S_q)$, which is denoted by $(\boldsymbol{\nu} \widetilde{sc})$ for short. Here, each s_{T_i} receives a channel over which it will send the necessary names: $s_{T_i} : *?(!(\widetilde{V_i}\widetilde{W_i}))$, where each $W_{i,j} = \overline{V_{i,j}}$. The channels s_{T_i} and c_{T_i} will also be referred to as s_j and c_j respectively.

5.4 Breakdown Function

We finally present the main definition of this thesis, which breaks a typable, replication-free process down into trios.

Definition 7 (*Trio breakdown function*)**.** Let P be a typable, replication-free process where $\Gamma \vdash P$. Let $k > 0$ and $\widetilde{u}$ be a sequence of pairwise distinct names. The Γ,k,$\widetilde{u}$-breakdown of P, written $\mathcal{B}^k_{\widetilde{u}}(P)$ is a process defined inductively as in Figure 9.

$\Gamma \vdash P$	$\mathcal{B}^k_{\widetilde{u}}(P)$	
$\Gamma \vdash \mathbf{0}$	$n_k(\widetilde{u})\,.\,\mathbf{0}$	
$\Gamma, y : T \vdash \overline{x_i}\langle y\rangle\,.\,Q$ let $\widetilde{u} = \widetilde{u_1} x_i \widetilde{u_2}$	if $T = \mathsf{bool}$ $n_k(\widetilde{u_1} x_i \widetilde{u_2})\,.\,\overline{x_i}\langle y\rangle\,.\,\overline{m_{k+1}}\langle \widetilde{u_1}\widetilde{u_2}\rangle\,.\,\mathbf{0}$ $\mid\ \mathcal{B}^{k+1}_{\widetilde{u_1}\widetilde{u_2}}(Q\{x_{i+1}\,/x_i\})$	
	otherwise $n_k(\widetilde{u_1} x_i \widetilde{u_2})\,.\,\overline{x_i}\langle \widetilde{y}\rangle\,.\,\overline{m_{k+1}}\langle \widetilde{v}\rangle\,.\,\mathbf{0}$ $\mid\ \mathcal{B}^{k+1}_{\widetilde{v}}(Q\{x_{i+1}\,/x_i\})$	$\widetilde{y} \subset \widetilde{u}, \quad \lvert\widetilde{y}\rvert = \mathrm{ord}(T)$ $\widetilde{v} = \widetilde{u_1}\widetilde{u_2} \setminus \widetilde{y}$
$\Gamma \vdash x_i(y : T)\,.\,Q$ let $\widetilde{u} = \widetilde{u_1} x_i \widetilde{u_2}$	$n_k(\widetilde{u_1} x_i \widetilde{u_2})\,.\,x_i(\widetilde{y} : \mathcal{G}(T))\,.\,\overline{m_{k+1}}\langle \widetilde{u_1}\widetilde{u_2}\widetilde{y}\rangle\,.\,\mathbf{0}$ $\mid\ \mathcal{B}^{k+1}_{\widetilde{u_1}\widetilde{u_2}\widetilde{y}}(Q\{x_{i+1}\,/x_i\}\{y_1\,/y\})$	$\lvert\widetilde{y}\rvert = \mathrm{ord}(T)$
$\Gamma \vdash (\boldsymbol{\nu} xy : T_j)Q$	$n_k(\widetilde{u})\,.\,(\boldsymbol{\nu} rt : \mathsf{lin}\,?(\widetilde{U}\widetilde{V}))\overline{c_j}\langle t\rangle\,.\,r(\widetilde{x}\widetilde{y} : \widetilde{U}\widetilde{V})\,.\,\overline{m_{k+1}}\langle \widetilde{u}\widetilde{x}\widetilde{y}\rangle\,.\,\mathbf{0}$ $\mid\ \mathcal{B}^{k+1}_{\widetilde{u}\widetilde{x}\widetilde{y}}(Q\{x_1\,/x\}\{y_1\,/y\})$	$\lvert\widetilde{x}\rvert = \lvert\widetilde{y}\rvert = \mathrm{ord}(T), \widetilde{U} = \mathcal{G}(T), V_i = \overline{U_i}$
$\Gamma_Q \circ \Gamma_R \vdash Q \mid R$	$n_k(\widetilde{u})\,.\,\overline{m_{k+1}}\langle \widetilde{u_Q}\rangle\,.\,\overline{m_l}\langle \widetilde{u_R}\rangle\,.\,\mathbf{0}$ $\mid\ \mathcal{B}^{k+1}_{\widetilde{u_Q}}(Q) \mid \mathcal{B}^{l}_{\widetilde{u_R}}(R)$	$\widetilde{u_Q} \subset \widetilde{u}$ and $\widetilde{u_R} \subset \widetilde{u}$ $l - k = \lvert Q\rvert + 1$

Figure 9: Breakdown Function Definition

We will show that the breakdown of a process as in Definition 7 is well-typed. For this, the following lemma will be useful, in which we use substitution of names on a sequence in a natural way.

Lemma 3 (*Typing preservation of trio breakdown under substitution of names*)**.** If $y \notin \mathtt{fn}(P)$ then

$$\Delta \vdash \mathcal{B}^k_{\widetilde{u}}(P) \implies \Delta \vdash \mathcal{B}^k_{\widetilde{u}\{y/x\}}(P\{y\ /x\}) \text{ for all } k, \widetilde{u}.$$

Proof. The proof is by induction on the structure of P. Let Γ be the context such that $\Gamma \vdash P$.

- The base case is trivial, since $\mathbf{0} = \mathbf{0}\{y\ /x\}$.
- For output, let $P = \alpha\ .\ Q$, we can distinguish three cases: two where $x \in \mathtt{fn}(\alpha)$, and one if $x \notin \alpha$. For $x \in \mathtt{fn}(\alpha)$, we can have $\alpha = \overline{x}\langle z\rangle$ for some z, or $\alpha = \overline{z}\langle x\rangle$ for some z:

  1. If $\alpha = \overline{x}\langle z\rangle$ for some z, then we know that $x \in \widetilde{u}$, so we write $\widetilde{u} = \widetilde{u_1} x \widetilde{u_2}$. Let $z : T \in \Gamma$. The arguments for the cases $T = \mathsf{bool}$ and $T \neq \mathsf{bool}$ are almost identical, so we show only $T = \mathsf{bool}$ here. The breakdown of P is given as:

  $$\mathcal{B}^k_{\widetilde{u_1}x\widetilde{u_2}}(P) = n_k(\widetilde{u_1}x\widetilde{u_2})\ .\ \overline{x}\langle z\rangle\ .\ \overline{m_{k+1}}\langle \widetilde{u_1}\widetilde{u_2}\rangle\ .\ \mathbf{0}\ \mid\ \mathcal{B}^{k+1}_{\widetilde{u_1}\widetilde{u_2}}(Q),$$

  and the lower part of the tree that proves that $\mathcal{B}^k_{\widetilde{u}}(P)$ is well-typed is

  $$\frac{\Delta_1 \vdash n_k(\widetilde{u_1}x\widetilde{u_2})\ .\ \overline{x}\langle z\rangle\ .\ \overline{m_{k+1}}\langle \widetilde{u_1}\widetilde{u_2}\rangle\ .\ \mathbf{0} \qquad \Delta_2 \vdash \mathcal{B}^{k+1}_{\widetilde{u_1}\widetilde{u_2}}(Q)}{\Delta = \Delta_1 \circ \Delta_2 \vdash n_k(\widetilde{u_1}x\widetilde{u_2})\ .\ \overline{x}\langle z\rangle\ .\ \overline{m_{k+1}}\langle \widetilde{u_1}\widetilde{u_2}\rangle\ .\ \mathbf{0}\ \mid\ \mathcal{B}^{k+1}_{\widetilde{u_1}\widetilde{u_2}}(Q)}\ [T\text{-}Par]. \tag{6}$$

  Both assumptions in this tree are well-justified, and we must prove that the breakdown of $P\{y\ /x\} = \overline{y}\langle z\rangle\ .\ Q\{y\ /x\}$ is also typable. The same rule is applied to construct the following proof tree:

  $$\frac{\Delta_1 \vdash n_k(\widetilde{u_1}y\widetilde{u_2})\ .\ \overline{y}\langle z\rangle\ .\ \overline{m_{k+1}}\langle \widetilde{u_1}\widetilde{u_2}\rangle\ .\ \mathbf{0} \qquad \dfrac{}{\Delta_2 \vdash \mathcal{B}^{k+1}_{\widetilde{u_1}\widetilde{u_2}\{y/x\}}(Q\{y\ /x\})}\ \text{I.H.}}{\Delta = \Delta_1 \circ \Delta_2 \vdash n_k(\widetilde{u_1}y\widetilde{u_2})\ .\ \overline{y}\langle z\rangle\ .\ \overline{m_{k+1}}\langle \widetilde{u_1}\widetilde{u_2}\rangle\ .\ \mathbf{0}\ \mid\ \mathcal{B}^{k+1}_{\widetilde{u_1}\widetilde{u_2}\{y/x\}}(Q\{y\ /x\})}\ [T\text{-}Par].$$

  The right assumption of this tree follows from the induction hypothesis (IH): if $y \notin \mathtt{fn}(Q)$ then $\Delta_2 \vdash \mathcal{B}^{k'}_{\widetilde{v}}(Q) \implies \Delta_2 \vdash \mathcal{B}^{k'}_{\widetilde{v}}(Q\{y\ /x\})$ for all $\widetilde{v}, k'$. In particular, $\widetilde{v} = \widetilde{u_1}\widetilde{u_2}\{y\ /x\}$ and $k' = k + 1$. Using α-conversion, the left assumption is also well-justified, because of (6).

  2. The case where $\alpha = \overline{z}\langle x\rangle$ for some channel z, and is shown in the same way as the previous case. Note that if $x : T \in \Gamma$ with $T \neq \mathsf{bool}$, then $\widetilde{x} \subset \widetilde{u}$, and we assume $\widetilde{y} \subset \widetilde{u}$.
  3. If $x \notin \alpha$, then $\alpha\{y\ /x\} = \alpha$, and the desired result follows immediately from the induction hypothesis.

- The case for input is completely analogous to the previous case.
- If $P = Q \mid R$, we have $P\{y\ /x\} = Q\{y\ /x\} \mid R\{y\ /x\}$ and after applying $[T\text{-}Par]$, we immediately apply the induction hypothesis twice on the breakdowns of Q and R.
- The case where $P = (\boldsymbol{\nu} wz)Q$ also follows readily from the induction hypothesis.

This concludes the proof that substitution on processes does not break typability of the trio breakdown. ☺

The following definition provides a notation for the context that types the trio breakdown of a process P.

Definition 8 (*Trio context*)**.** Let Γ be a context. The *trio context* of Γ, written Δ_Γ, is defined as

$$\Delta_\Gamma = \mathcal{U}(\Gamma),\quad \bigcup_{i\in\{k,\ldots,k+h\}} n_i : \mathsf{lin}\ ?\widetilde{U_i},\quad \bigcup_{i\in\{k+1,\ldots,k+h\}} m_i : \mathsf{lin}\ !\widetilde{U_i},\quad \bigcup_{i\in\{1,\ldots,q\}} c_i : *!(!(\widetilde{V_i}\widetilde{W_i})), \tag{7}$$

where $q, h \geq 0$, and $\widetilde{U_k}, \ldots, \widetilde{U_{k+l}}$, $\widetilde{V_1}, \ldots, \widetilde{V_q}$ and $\widetilde{W_1}, \ldots, \widetilde{W_q}$ are given lists of minimal types such that $W_{i,j} = \overline{V_{i,j}}$ for each $i \in \{1, \ldots, q\}$ and $j \in \{1, \ldots, |V_i|\}$.

In the following lemma, $h = |P| - 1$, and q is the number of distinct types T_i such that $(\boldsymbol{\nu} xy : T_i)$ occurs in P, and we take $\widetilde{V_i} = \mathcal{G}(T_i)$. Furthermore, each $\widetilde{U_i}$ is derived from one of the rules in Figure 9, and they will be made explicit in the proof.

Lemma 4 (*Typability of the trio breakdown*)**.** If $\Gamma \vdash P$ then there exist lists of minimal types such that $\Delta_\Gamma \vdash \mathcal{B}^k_{\widetilde{u}}(P)$, for any k and $\widetilde{u}$.

Proof. The proof is by induction on the structure of P, providing the list of minimal types $\widetilde{U_k}$ for $\widetilde{u}$ at each inductive step. We will omit the first assumption for Rule $[T\text{-}In]$ (8) for linear types, namely $\mathsf{lin}\ (\Gamma_1 \circ \Gamma_2)$, since it is always satisfied. The collection of judgments for the unrestricted channels c_i for $i \in \{1, \ldots, q\}$ will be denoted by Ξ_q, and we sometimes write Θ to denote their union $\mathcal{U}(\Gamma), \Xi_q$.

$$\Xi_q = \bigcup_{i\in\{1,\ldots,q\}} c_i : *!(!(\widetilde{V_i}\widetilde{W_i})), \qquad \Theta = \mathcal{U}(\Gamma), \Xi_q \tag{8}$$

Note that now Θ only contains judgments of unrestricted types. We apply unrestricted weakening [12] while typing a trio, explicitly or silently. For ease of reference, we formulate the induction hypothesis as follows:

$$\text{If } \Gamma' \vdash P' \text{ then } \Delta_{\Gamma'} \vdash \mathcal{B}^{k'}_{\widetilde{u'}}(P'),\ \text{for any } k', \widetilde{u'} \tag{9}$$

- The base case, where $P = \mathbf{0}$ and $|P| = 1$, we have $h = 0$ and $q = 0$. Then for any Γ such that $\Gamma \vdash \mathbf{0}$, we have $\Delta_\Gamma = \mathcal{U}(\Gamma), n_k : \mathsf{lin}?\widetilde{U_i}.\mathsf{end}$ by Definition 8, and the following proof tree shows that the trio for termination is typed by Δ_Γ:

$$\dfrac{\dfrac{\mathsf{un}(\mathcal{U}(\Gamma))}{\mathcal{U}(\Gamma), n_k : \mathsf{lin}?\widetilde{U_k}.\mathsf{end} \vdash n_k : \mathsf{lin}?\widetilde{U_k}.\mathsf{end}}\,[T\text{-}Var] \qquad \dfrac{\mathsf{un}(\mathcal{U}(\Gamma), n_k : \mathsf{end}, \widetilde{u} : \widetilde{U_k})}{\mathcal{U}(\Gamma), n_k : \mathsf{end}, \widetilde{u} : \widetilde{U_k} \vdash \mathbf{0}}\,[T\text{-}Inact]}{\mathcal{U}(\Gamma), n_k : \mathsf{lin}?\widetilde{U_k}.\mathsf{end} \vdash n_k(\widetilde{u})\ .\ \mathbf{0}}\,[T\text{-}In]$$

  The top left assumption is trivially satisfied. We may assume that for all judgments $u_i : T \in \widetilde{u} : \widetilde{U_k}$, either $T = \mathsf{bool}$, or $T = \mathsf{end}$, since $P = \mathbf{0}$, and all sessions have ended. Consequently, the top right assumption is also satisfied.

- For output, we write $P = \overline{x_i}\langle y\rangle\ .\ Q$ and $\Gamma \vdash P$, and we first show the start of the proof tree for P:

$$\dfrac{\Gamma_1 \vdash x_i : !T.U \qquad \Gamma_2 \vdash y : T \qquad \Gamma_3, x_i : U \vdash Q}{\Gamma \vdash \overline{x_i}\langle y\rangle\ .\ Q}\,[T\text{-}Out] \tag{10}$$

where $\Gamma = \Gamma_1 \circ \Gamma_2 \circ \Gamma_3$. We can distinguish two subcases:

1. Subcase $T = \mathsf{bool}$: we will build the proof tree for the trio breakdown of P. The only rule that can be applied is $[T\text{-}Par]$.

$$\frac{\Delta_\Gamma \vdash n_k(\widetilde{u_1}x_i\widetilde{u_2}) \,.\, \overline{x_i}\langle y\rangle \,.\, \overline{m_{k+1}}\langle\widetilde{u_1}\widetilde{u_2}\rangle \,.\, \mathbf{0} \mid \mathcal{B}^{k+1}_{\widetilde{u_1}\widetilde{u_2}}(Q\{x_{i+1}\,/x_i\})}{\dfrac{\dfrac{\dfrac{}{\Delta_{\Gamma'} \vdash \mathcal{B}^{k+1}_{\widetilde{u_1}\widetilde{u_2}\{x_i/x_{i+1}\}}(Q)}\,\text{I.H.}}{\Delta_{\Gamma'} \vdash \mathcal{B}^{k+1}_{\widetilde{u_1}\widetilde{u_2}}(Q\{x_{i+1}\,/x_i\})}\,\textsc{Lemma 3}}{\Delta_\Gamma \vdash n_k(\widetilde{u_1}x_i\widetilde{u_2}) \,.\, \overline{x_i}\langle y\rangle \,.\, \overline{m_{k+1}}\langle\widetilde{u_1}\widetilde{u_2}\rangle \,.\, \mathbf{0} \mid \mathcal{B}^{k+1}_{\widetilde{u_1}\widetilde{u_2}}(Q\{x_{i+1}\,/x_i\})}\,[T\text{-}Par]} \tag{11}$$

(12)

The induction hypothesis (I.H.) (9) is applied with $P' = Q$ and $\Gamma' := \Gamma_3, x_i : U$. In particular, we use $k' = k + 1$, and $\widetilde{u'} = \widetilde{u_1}\widetilde{u_2}\{x_i\ /x_{i+1}\}$. We also apply Lemma 3 for the right-hand side assumption. Specifically, by the I.H., $\Delta_{\Gamma'} \vdash \mathcal{B}^{k+1}_{\widetilde{u_1}\widetilde{u_2}\{x_i/x_{i+1}\}}(Q)$, and then by Lemma 3: $\Delta_{\Gamma'} \vdash \mathcal{B}^{k+1}_{\widetilde{u_1}\widetilde{u_2}}(Q\{x_{i+1}\ /x_i\})$, where the inverse substitution makes sure that $\widetilde{u_1}\widetilde{u_2}\{x_i\ /x_{i+1}\}\{x_{i+1}\ /x_i\} = \widetilde{u_1}\widetilde{u_2}$. In this lemma, we have assumed that $x_{i+1} \in \widetilde{u_1}\widetilde{u_2}$, which holds, unless x_i is the last action in the session. In that case $x_i \notin \mathtt{fn}(Q)$, and the substitution on Q has no effect. In that case the I.H. can be applied immediately without Lemma 3.

The subtree on the left-hand side is provided below, where the reader should note that either $y \in \widetilde{u_1}\widetilde{u_2}$ or not, but always $y : \mathsf{bool} \in \mathcal{U}(\Gamma)$. Since the type is unrestricted, the tree is well-justified either way.

$$\frac{\dfrac{}{n_k :?\widetilde{U_k} \vdash n_k :?\widetilde{U_k}} \quad \dfrac{\dfrac{}{x_i :!\mathsf{bool} \vdash x_i :!\mathsf{bool}} \quad \dfrac{}{y : \mathsf{bool} \vdash y : \mathsf{bool}}\ (13)}{\Theta, m_{k+1} :!\widetilde{U_{k+1}}, \widetilde{u_1}x_i\widetilde{u_2} : \widetilde{U_k} \vdash \overline{x_i}\langle y\rangle \,.\, \overline{m_{k+1}}\langle\widetilde{u_1}\widetilde{u_2}\rangle \,.\, \mathbf{0}}\,[T\text{-}Out]}{\Theta, n_k :?\widetilde{U_k}, m_{k+1} :!\widetilde{U_{k+1}} \vdash n_k(\widetilde{u_1}x_i\widetilde{u_2}) \,.\, \overline{x_i}\langle y\rangle \,.\, \overline{m_{k+1}}\langle\widetilde{u_1}\widetilde{u_2}\rangle \,.\, \mathbf{0}}\,[T\text{-}In] \tag{12}$$

$$\frac{\dfrac{}{m_{k+1} :!\widetilde{U_{k+1}} \vdash m_{k+1} :!\widetilde{V}} \qquad \dfrac{}{\widetilde{u_1}\widetilde{u_2} : \widetilde{V} \vdash \widetilde{u_1}\widetilde{u_2} : \widetilde{V}} \qquad \dfrac{}{\emptyset \vdash \mathbf{0}}}{\widetilde{u_1}\widetilde{u_2} : \widetilde{V}, m_{k+1} :!\widetilde{U_{k+1}} \vdash \overline{m_{k+1}}\langle\widetilde{u_1}\widetilde{u_2}\rangle \,.\, \mathbf{0}}\,[T\text{-}Out] \tag{13}$$

The left assumption in (13) is satisfied by picking $\widetilde{U_{k+1}} = \widetilde{V}$. The middle assumption is well-justified by a generalization of Rule $[T\text{-}Var]$ for polyadicity. In the right assumption, we applied unrestricted weakening on $\mathcal{U}(\widetilde{u_1}\widetilde{u_2} : \widetilde{V})$. Finally, by Definition 8, in (11) we have:

$$\Delta_{\Gamma'} = \Theta, \bigcup_{i\in\{k+1,\ldots,k+l\}} n_i : \mathsf{lin}\ ?\widetilde{U_i}, \bigcup_{i\in\{k+2,\ldots,k+l\}} m_i : \mathsf{lin}\ !\widetilde{U_i}$$

$$\Delta_\Gamma = \Theta, \bigcup_{i\in\{k,\ldots,k+l\}} n_i : \mathsf{lin}\ ?\widetilde{U_i}, \bigcup_{i\in\{k+1,\ldots,k+l\}} m_i : \mathsf{lin}\ !\widetilde{U_i}$$

Also, by definition of context split, we have:

$$\Delta_\Gamma = (\Theta, n_k : \mathsf{lin}\ ?\widetilde{U_k}, m_{k+1} : \mathsf{lin}\ !\widetilde{U_{k+1}}) \circ \Delta_{\Gamma'}, \tag{14}$$

which concludes the proof for the inductive step in this subcase.

2. Subcase $T \neq \mathsf{bool}$: we use the same reasoning as in the previous subcase. After applying Rule $[T\text{-}Par]$, the I.H. and Lemma 3 are applied similarly, but now taking $\widetilde{u}' = \widetilde{v}$, where $\widetilde{v} = \widetilde{u_1}\widetilde{u_2} \setminus \widetilde{y}$. The trio is also different, and we show its typing here. Note that by construction we have $\widetilde{y} \subset \widetilde{u_1}\widetilde{u_2}$.

$$\dfrac{\dfrac{}{n_k : ?\widetilde{U_k} \vdash n_k : ?\widetilde{U_k}} \qquad \dfrac{\dfrac{}{x_i : !\widetilde{M} \vdash x_i : !\widetilde{M}} \qquad \dfrac{}{\widetilde{y} : \widetilde{M} \vdash \widetilde{y} : \widetilde{M}} \qquad (15)}{n_k : \mathsf{end}, m_{k+1} : !\widetilde{U_{k+1}}, \widetilde{u_1} x_i \widetilde{u_2} : \widetilde{U_k} \vdash \overline{x_i}\langle\widetilde{y}\rangle \,.\, \overline{m_{k+1}}\langle\widetilde{v}\rangle \,.\, \mathbf{0}}\,[T\text{-}Out]}{n_k : ?\widetilde{U_k}, m_{k+1} : !\widetilde{U_{k+1}} \vdash n_k(\widetilde{u_1} x_i \widetilde{u_2}) \,.\, \overline{x_i}\langle\widetilde{y}\rangle \,.\, \overline{m_{k+1}}\langle\widetilde{v}\rangle \,.\, \mathbf{0}}\,[T\text{-}In]$$

All names of $\widetilde{y}$ are linear, so we continue (15) with $\widetilde{v} = \widetilde{u_1}\widetilde{u_2} \setminus \widetilde{y}$. The tree is finalized as follows:

$$\dfrac{\dfrac{}{m_{k+1} : !\widetilde{U_{k+1}} \vdash m_{k+1} : !\widetilde{V}} \qquad \dfrac{}{\widetilde{v} : \widetilde{V} \vdash \widetilde{v} : \widetilde{V}} \qquad \dfrac{}{\emptyset \vdash \mathbf{0}}}{x_i : \mathsf{end}, m_{k+1} : !\widetilde{U_{k+1}}, \widetilde{v} : \widetilde{V} \vdash \overline{m_{k+1}}\langle\widetilde{v}\rangle \,.\, \mathbf{0}}\,[T\text{-}Out] \qquad (15)$$

To justify the leftmost assumption in (15), we pick $\widetilde{U_{k+1}} = \widetilde{V}$. The middle assumption is proven with a generalization of Rule $[T\text{-}Var]$ for polyadicity. In conclusion, Δ_Γ is constructed identical to the previous subcase (14).

These two subcases conclude the proof for the case of output.

- For the case of input, we write $P = x_i(y : T) \,.\, Q$, and the only rule that can be applied is $[T\text{-}In]$, which we show here.

$$\dfrac{\Gamma_1 \vdash x_i : ?T.U \qquad \Gamma_2, x_i : U, y : T \vdash Q}{\Gamma \vdash x_i(y : T) \,.\, Q}\,[T\text{-}In]$$

where $\Gamma = \Gamma_1 \circ \Gamma_2$. Like in the previous case, we set $\Gamma' = \Gamma_2, x_i : U, y : T$. Then, we apply the I.H. with $k' = k + 1$ and $\widetilde{u}' = \widetilde{u_1}\widetilde{u_2}\{x_i \,/x_{i+1}\}\widetilde{y}\{y \,/y_1\}$. The two substitutions here are inversions of the substitutions applied by Lemma 3. Then, applying the lemma twice on the breakdown of the continuation, we arrive at the following tree:

$$(16)\quad \dfrac{\Delta_\Gamma \vdash n_k(\widetilde{u_1} x_i \widetilde{u_2}) \,.\, x_i(\widetilde{y}) \,.\, \overline{m_{k+1}}\langle\widetilde{u_1}\widetilde{u_2}\widetilde{y}\rangle \,.\, \mathbf{0} \mid \mathcal{B}^{k+1}_{\widetilde{u_1}\widetilde{u_2}\widetilde{y}}(Q\{x_{i+1} \,/x_i\}\{y_1 \,/y\})}{\dfrac{\dfrac{\dfrac{}{\Delta_{\Gamma'} \vdash \mathcal{B}^{k+1}_{\widetilde{u_1}\widetilde{u_2}\{x_i/x_{i+1}\}\widetilde{y}\{y/y_1\}}(Q)}\,\text{I.H.}}{\Delta_{\Gamma'} \vdash \mathcal{B}^{k+1}_{\widetilde{u_1}\widetilde{u_2}\widetilde{y}\{y/y_1\}}(Q\{x_{i+1} \,/x_i\})}\,\textsc{Lemma 3}}{\Delta_{\Gamma'} \vdash \mathcal{B}^{k+1}_{\widetilde{u_1}\widetilde{u_2}\widetilde{y}}(Q\{x_{i+1} \,/x_i\}\{y_1 \,/y\})}\,\textsc{Lemma 3}}\,[T\text{-}Par]$$

We prove that the trio in (16) is typed as follows:

$$\dfrac{\dfrac{}{n_k : \widetilde{U_k} \vdash n_k : ?\widetilde{U_k}} \qquad \dfrac{\dfrac{}{x_i : ?\widetilde{M} \vdash x_i : ?\widetilde{M}} \qquad (17)}{n_k : \mathsf{end}, m_{k+1} : !\widetilde{U_{k+1}}, \widetilde{u_1} x_i \widetilde{u_2} : \widetilde{U_k} \vdash x_i(\widetilde{y}) \,.\, \overline{m_{k+1}}\langle\widetilde{u_1}\widetilde{u_2}\widetilde{y}\rangle \,.\, \mathbf{0}}\,[T\text{-}In]}{n_k : ?\widetilde{U_k}, m_{k+1} : !\widetilde{U_{k+1}} \vdash n_k(\widetilde{u_1} x_i \widetilde{u_2}) \,.\, x_i(\widetilde{y}) \,.\, \overline{m_{k+1}}\langle\widetilde{u_1}\widetilde{u_2}\widetilde{y}\rangle \,.\, \mathbf{0}}\,[T\text{-}In] \qquad (16)$$

Here, we have $\widetilde{M} = \mathcal{G}(T)$. We continue building the proof tree on the right, where we first assume that $T \neq \mathsf{bool}$.

$$\frac{\overline{m_{k+1} :!\widetilde{U_{k+1}} \vdash m_{k+1} :!\widetilde{U_{k+1}}} \qquad \overline{\widetilde{u_1}\widetilde{u_2}\widetilde{y} : \widetilde{V}\widetilde{M} \vdash \widetilde{u_1}\widetilde{u_2}\widetilde{y} : \widetilde{V}\widetilde{M}} \qquad \overline{\emptyset \vdash \mathbf{0}}}{x_i : \mathsf{end}, m_{k+1} :!\widetilde{U_{k+1}}, \widetilde{u_1}\widetilde{u_2} : \widetilde{V}, \widetilde{y} : \widetilde{M} \vdash \overline{m_{k+1}}\langle \widetilde{u_1}\widetilde{u_2}\widetilde{y}\rangle \,.\, \mathbf{0}}\,[T\text{-}Out] \tag{17}$$

All assumptions are justified similarly to the case for output, and Δ_Γ is constructed identically. This concludes the proof of the induction step for input.

- Here, we prove the inductive step for $P = (\boldsymbol{\nu} xy : T_j)Q$. In the previous two steps, the emphasis was on the judgments for n_k and m_{k+1}, which are used here similarly. Here we will also use Ξ_q in (8). Specifically, we will need the judgment for the unrestricted channel c_j to type the trio. First, we show the last rule applied to show that $\Gamma \vdash P$.

$$\frac{\Gamma, x : T, y : \overline{T} \vdash Q}{\Gamma \vdash (\boldsymbol{\nu} xy)Q}\,[T\text{-}Res] \tag{18}$$

We let $\Gamma' = \Gamma, x : T, y : \overline{T}$. Now, we show that the trio breakdown of P is well-typed. The only rule we can apply is $[T\text{-}Par]$. The I.H. is applied with $P' = Q$ and using $k' = k + 1$ and $\widetilde{u'} = \widetilde{u}\widetilde{x}\{^x/_{x_1}\}\widetilde{y}\{^y/_{y_1}\}$. For the lemma, the inverse substitutions are used similarly to the case for input.

$$\frac{\dfrac{\dfrac{}{\Delta_{\Gamma'} \vdash \mathcal{B}^{k+1}_{\widetilde{u}\widetilde{x}\{x/x_1\}\widetilde{y}\{y/y_1\}}(Q)}\,\text{I.H.}}{\Delta_{\Gamma'} \vdash \mathcal{B}^{k+1}_{\widetilde{u}\widetilde{x}\widetilde{y}}(Q\{x_1\,/x\}\{y_1\,/y\})}\,2 \times \textsc{Lemma 3}}{\Delta_\Gamma \vdash n_k(\widetilde{u}) \,.\, (\boldsymbol{\nu} rt)\overline{c_j}\langle t\rangle \,.\, r(\widetilde{x}\widetilde{y}) \,.\, \overline{m_{k+1}}\langle \widetilde{u}\widetilde{x}\widetilde{y}\rangle \,.\, \mathbf{0} \mid \mathcal{B}^{k+1}_{\widetilde{u}\widetilde{x}\widetilde{y}}(Q\{x_1\,/x\}\{y_1\,/y\})}\,[T\text{-}Par] \tag{20}$$

(19)

We have applied unrestricted weakening on all judgments in Θ, except for c_j.

$$\frac{\overline{n_k :?\widetilde{U_k} \vdash n_k :?\widetilde{U_k}} \qquad \dfrac{(21)}{c_j : *!(!(\widetilde{V_j}\widetilde{W_j})), m_{k+1} :!\widetilde{U_{k+1}}, \widetilde{u} : \widetilde{U_k} \vdash (\boldsymbol{\nu} rt)\overline{c_j}\langle t\rangle \,.\, r(\widetilde{x}\widetilde{y}) \,.\, \overline{m_{k+1}}\langle \widetilde{u}\widetilde{x}\widetilde{y}\rangle \,.\, \mathbf{0}}\,[T\text{-}Res]}{c_j : *!(!(\widetilde{V_j}\widetilde{W_j})), n_k :?\widetilde{U_k}, m_{k+1} :!\widetilde{U_{k+1}} \vdash n_k(\widetilde{u}) \,.\, (\boldsymbol{\nu} rt)\overline{c_j}\langle t\rangle \,.\, r(\widetilde{x}\widetilde{y}) \,.\, \overline{m_{k+1}}\langle \widetilde{u}\widetilde{x}\widetilde{y}\rangle \,.\, \mathbf{0}}\,[T\text{-}In] \tag{20}$$

$$\frac{\overline{c_j : *!(!(\widetilde{V_j}\widetilde{W_j})) \vdash c_j : *!(!(\widetilde{V_j}\widetilde{W_j}))} \qquad \overline{t :!(\widetilde{V_j}\widetilde{W_j}) \vdash t :!(\widetilde{V_j}\widetilde{W_j})} \qquad (22)}{c_j : *!(!(\widetilde{V_j}\widetilde{W_j})), m_{k+1} :!\widetilde{U_{k+1}}, \widetilde{u} : \widetilde{U_k}, r :?(\widetilde{V_j}\widetilde{W_j}), t :!(\widetilde{V_j}\widetilde{W_j}) \vdash \overline{c_j}\langle t\rangle \,.\, r(\widetilde{x}\widetilde{y}) \,.\, \overline{m_{k+1}}\langle \widetilde{u}\widetilde{x}\widetilde{y}\rangle \,.\, \mathbf{0}}\,[T\text{-}Out] \tag{21}$$

$$\frac{\overline{r :?(\widetilde{V_j}\widetilde{W_j}) \vdash r :?(\widetilde{V_j}\widetilde{W_j})} \quad m_{k+1} :!\widetilde{U_{k+1}}, \widetilde{u} : \widetilde{U_k}, \widetilde{x} : \widetilde{V_j}, \widetilde{y} : \widetilde{W_j} \vdash \overline{m_{k+1}}\langle \widetilde{u}\widetilde{x}\widetilde{y}\rangle \,.\, \mathbf{0}}{m_{k+1} :!\widetilde{U_{k+1}}, \widetilde{u} : \widetilde{U_k}, r :?(\widetilde{V_j}\widetilde{W_j}) \vdash r(\widetilde{x}\widetilde{y}) \,.\, \overline{m_{k+1}}\langle \widetilde{u}\widetilde{x}\widetilde{y}\rangle \,.\, \mathbf{0}}\,[T\text{-}In] \tag{22}$$

The proof tree can be finished by choosing $\widetilde{U_{k+1}} = \widetilde{U_k}\widetilde{V_j}\widetilde{W_j}$, similarly to the case for input (17). From (19) and (20) we get:

$$\Delta_\Gamma = (c_j : *!(!(\widetilde{V_j}\widetilde{W_j})), n_k : \mathsf{lin}\ ?\widetilde{U_k}, m_{k+1} : \mathsf{lin}\ !\widetilde{U_{k+1}}) \circ \Delta_{\Gamma'},$$

and with that, the proof of the inductive step for restriction is finished.

- Finally, we prove the case for parallel composition. We write $P = Q \mid R$, and $\Gamma = \Gamma_1 \circ \Gamma_2$ in the following tree:

$$\frac{\Gamma_1 \vdash Q \qquad \Gamma_2 \vdash R}{\Gamma_1 \circ \Gamma_2 \vdash Q \mid R}\,[T\text{-}Par]$$

Here, both Rule $[T\text{-}Par]$ and the I.H. are applied twice. The I.H. is used with $P' = Q$ and $\Gamma' = \Gamma_Q$, in particular for $k' = k+1$ and $\widetilde{u'} = \widetilde{u_Q}$ and similarly for $P' = R$ with $k' = l$. Recall that $l = k + |Q| + 1$ by construction, so we know that $m_l : \mathsf{lin}\ ! : \widetilde{U_l} \in \Delta_\Gamma$, because $l \leq k + h$, since the number of subterms in P is at least the number of subterms in Q. The following proof tree shows that the breakdown of parallel composition can be typed:

$$\frac{\dfrac{(24) \qquad \dfrac{}{\Delta_{\Gamma_1} \vdash \mathcal{B}^{k+1}_{\widetilde{u_Q}}(Q)}\,\text{I.H.}}{\Delta' \vdash n_k(\widetilde{u}) \,.\, \overline{m_{k+1}}\langle\widetilde{u_Q}\rangle \,.\, \overline{m_l}\langle\widetilde{u_R}\rangle \,.\, \mathbf{0} \mid \mathcal{B}^{k+1}_{\widetilde{u_Q}}(Q)}\,[T\text{-}Par] \quad \dfrac{}{\Delta_{\Gamma_2} \vdash \mathcal{B}^{l}_{\widetilde{u_R}}(R)}\,\text{I.H.}}{\Delta_\Gamma \vdash n_k(\widetilde{u}) \,.\, \overline{m_{k+1}}\langle\widetilde{u_Q}\rangle \,.\, \overline{m_l}\langle\widetilde{u_R}\rangle \,.\, \mathbf{0} \mid \mathcal{B}^{k+1}_{\widetilde{u_Q}}(Q) \mid \mathcal{B}^{l}_{\widetilde{u_R}}(R)}\,[T\text{-}Par] \tag{23}$$

Here, $\Delta_\Gamma = \Delta' \circ \Delta_{\Gamma_2}$ and $\Delta' = \Delta'' \circ \Delta_{\Gamma_1}$, where Δ'' is the context in the base of the tree that types the trio:

$$\frac{\dfrac{}{n_k :?\widetilde{U_k} \vdash n_k :?\widetilde{U_k}} \quad \dfrac{\dfrac{}{m_{k+1} :!\widetilde{U_{k+1}} \vdash m_{k+1} :!\widetilde{V_Q}} \quad \dfrac{}{\widetilde{u_Q} : \widetilde{V_Q}} \qquad (25)}{m_{k+1} :!\widetilde{U_{k+1}}, m_l :!\widetilde{U_l}, \widetilde{u} : \widetilde{U_k} \vdash \overline{m_{k+1}}\langle\widetilde{u_Q}\rangle \,.\, \overline{m_l}\langle\widetilde{u_R}\rangle \,.\, \mathbf{0}}\,[T\text{-}Out]}{\Delta'' = n_k :?\widetilde{U_k}, m_{k+1} :!\widetilde{U_{k+1}}, m_l :!\widetilde{U_l} \vdash n_k(\widetilde{u}) \,.\, \overline{m_{k+1}}\langle\widetilde{u_Q}\rangle \,.\, \overline{m_l}\langle\widetilde{u_R}\rangle \,.\, \mathbf{0}}\,[T\text{-}In] \tag{24}$$

Recall that by construction (5.2), $\widetilde{u_Q}$ does not contain linear names that are also in $\widetilde{u_R}$. More specifically, $(\widetilde{u_Q} : \widetilde{V_Q}) \circ (\widetilde{u_R} : \widetilde{V_R}) = \widetilde{u} : \widetilde{U_k}$, where $\widetilde{V_R}$ is as in the following subtree:

$$\frac{\dfrac{}{m_l :!\widetilde{U_l} \vdash m_l :!\widetilde{V_R}} \quad \dfrac{}{\widetilde{u_R} : \widetilde{V_R} \vdash \widetilde{u_R} : \widetilde{V_R}} \quad \dfrac{}{\emptyset \vdash \mathbf{0}}}{m_l :!\widetilde{U_l}, \widetilde{u_R} : \widetilde{V_R} \vdash \overline{m_l}\langle\widetilde{u_R}\rangle \,.\, \mathbf{0}}\,[T\text{-}Out] \tag{25}$$

From (23) we have $\Delta_\Gamma = \Delta'' \circ \Delta_{\Gamma_1} \circ \Delta_{\Gamma_2}$, where

$$\begin{aligned}
\Delta_{\Gamma_1} = \;& \Theta, \bigcup_{i \in \{k+1,\ldots,l-1\}} n_i : \mathsf{lin}\ ?\widetilde{U_i}, \bigcup_{i \in \{k+2,\ldots,l-1\}} m_i : \mathsf{lin}\ !\widetilde{U_i}, \\
\Delta_{\Gamma_2} = \;& \Theta, \bigcup_{i \in \{l,\ldots,k+h\}} n_i : \mathsf{lin}\ ?\widetilde{U_i}, \bigcup_{i \in \{l+1,\ldots,k+h\}} m_i : \mathsf{lin}\ !\widetilde{U_i},
\end{aligned}$$

and $k + 1 < l < k + h$. The proof is thus concluded.

☺

6 Main Result

The breakdown function creates a process of parallel trios, but still needs to be activated if it is to simulate the original process. Also, we need restrictions to make sure that every m_k synchronizes with n_k. Finally, the name inventor with restrictions for every c_j communicating with s_j needs to be added. This motivates the following definition, which can be considered as a typed variant of the definition by Parrow [7].

Definition 9 (*Trio translation*)**.** Let P be a process such that $\Gamma \vdash P$, where Γ is a value environment. The *trio translation* of P, denoted by $\mathcal{T}(P)$, is defined as

$$\mathcal{T}(P) = (\boldsymbol{\nu}\widetilde{m}\widetilde{n})(\boldsymbol{\nu}\widetilde{sc})(\overline{m_1}\langle\rangle \,.\, \mathbf{0} \mid \mathcal{B}^1_\epsilon(P) \mid N_P)$$

where N_P is the name inventor as in Definition 6 and $\mathcal{B}^k_{\widetilde{u}}(P)$ is the Γ,k,$\widetilde{u}$-breakdown function from Definition 7. Furthermore, $|\widetilde{m}| = |\widetilde{n}| = |P|$, and $|\widetilde{s}| = |\widetilde{c}|$ is the number of types T such that $(\boldsymbol{\nu}xy : T)$ occurs in P.

It can be safely assumed that our own invented names m_k and n_k are not in conflict with any names in P or $\mathcal{B}^1_\epsilon(P)$ by alpha-conversion, and similarly for the channels c_j and s_j. With this definition and Lemma 4, we easily prove the main theorem of this thesis.

Theorem 1 (*Typability of the trio translation*)**.** Let P and $\mathcal{T}(P)$ be as in Definition 9.

$$\text{If } \Gamma \vdash P \text{ then } \Gamma \vdash \mathcal{T}(P)$$

Proof. By Definition 9, $x : T \in \Gamma$ implies $T = \mathsf{bool}$, so we have $\mathsf{un}(\Gamma)$. Therefore, $\Gamma = \mathcal{U}(\Gamma)$, and we prove that $\mathcal{U}(\Gamma) \vdash \mathcal{T}(P)$. Let $h = |P|$ and q be the number of types T such that $(\boldsymbol{\nu}xy : T)$ occurs in P. We first type all outer restrictions on the translation as follows:

$$\dfrac{\dfrac{\dfrac{\mathcal{U}(\Gamma), \widetilde{m} : \widetilde{M}, \widetilde{n} : \widetilde{N}, \bigcup_{i\in\{1,\ldots,q\}} c_i : *!(!(\widetilde{V_i}\widetilde{W_i})), \bigcup_{i\in\{1,\ldots,q\}} s_i : *?(!(\widetilde{V_i}\widetilde{W_i})) \vdash \overline{m_1}\langle\rangle \,.\, \mathbf{0} \mid \mathcal{B}^1_\epsilon(P) \mid N}{\vdots\,(q)}\,[T\text{-}Res]}{\mathcal{U}(\Gamma), m_1 : !\mathsf{end}, n_1 : ?\mathsf{end}, \ldots, m_h : !\widetilde{U_h}, n_h : ?\widetilde{U_h} \vdash (\boldsymbol{\nu}\widetilde{sc})(\overline{m_1}\langle\rangle \,.\, \mathbf{0} \mid \mathcal{B}^1_\epsilon(P) \mid N)}\,[T\text{-}Res]}{\vdots\,(h)}\,[T\text{-}Res]}{\mathcal{U}(\Gamma) \vdash (\boldsymbol{\nu}\widetilde{m}\widetilde{n})(\boldsymbol{\nu}\widetilde{sc})(\overline{m_1}\langle\rangle \,.\, \mathbf{0} \mid \mathcal{B}^1_\epsilon(P) \mid N)}\,[T\text{-}Res] \tag{26}$$

Naturally, the channels m_1 and n_1 communicate nothing. Each m_k for $k \geq 2$ outputs a sequence with a list of types as made explicit in the proof of Lemma 4, and we choose n_k to be its dual endpoint. We write the collection of judgments for $m_1, \ldots, m_h$ and $n_1, \ldots, n_h$ as $\widetilde{m} : \widetilde{M}$ and $\widetilde{n} : \widetilde{N}$, and continue the proof tree (26) as follows:

Next, we apply Rule $[T\text{-}Par]$ twice, while using unrestricted weakening. This leads to the following continuation of the tree in (26):

$$\dfrac{\dfrac{}{m_1 : !\mathsf{end} \vdash \overline{m_1}\langle\rangle \,.\, \mathbf{0}} \quad \dfrac{}{\Delta_\Gamma \vdash \mathcal{B}^1_\epsilon(P)}\,\textsc{Lemma 4}}{\mathcal{U}(\Gamma), \widetilde{m} : \widetilde{M}, \widetilde{n} : \widetilde{N}, \bigcup_{i\in\{1,\ldots,q\}} c_i : *!(!(\widetilde{V_i}\widetilde{W_i})) \vdash \overline{m_1}\langle\rangle \,.\, \mathbf{0} \mid \mathcal{B}^1_\epsilon(P)}\,[T\text{-}Par] \tag{28}$$

$$\dfrac{}{\mathcal{U}(\Gamma), \widetilde{m} : \widetilde{M}, \widetilde{n} : \widetilde{N}, \bigcup_{i\in\{1,\ldots,q\}} c_i : *!(!(\widetilde{V_i}\widetilde{W_i})), \bigcup_{i\in\{1,\ldots,q\}} s_i : *?(!(\widetilde{V_i}\widetilde{W_i})) \vdash \overline{m_1}\langle\rangle \,.\, \mathbf{0} \mid \mathcal{B}^1_\epsilon(P) \mid N}\,[T\text{-}Par] \tag{27}$$

What is left to show is that the name inventor is well-typed.

$$\dfrac{(29) \qquad s_2 : *?(!(\widetilde{V_2}\widetilde{W_2})), \ldots, s_q : *?(!(\widetilde{V_q}\widetilde{W_q})) \vdash N_{T_2} \mid \cdots \mid N_{T_q}}{s_1 : *?(!(\widetilde{V_1}\widetilde{W_1})), s_2 : *?(!(\widetilde{V_2}\widetilde{W_2})), \ldots, s_q : *?(!(\widetilde{V_q}\widetilde{W_q})) \vdash N_{T_1} \mid N_{T_2} \mid \cdots \mid N_{T_q}}\,[T\text{-}Par] \tag{28}$$

The first name server is typed in (29):

$$\dfrac{\mathsf{un}(s_1 : *?(!(\widetilde{V_1}\widetilde{W_1}))) \qquad s_1 : \mathsf{un}\ ?(!(\widetilde{V_1}\widetilde{W_1})).*?(!(\widetilde{V_1}\widetilde{W_1}))) \vdash s_1 : \mathsf{un}\ ?(!(\widetilde{V_1}\widetilde{W_1})).*?(!(\widetilde{V_1}\widetilde{W_1})))}{s_1 : *?(!(\widetilde{V_1}\widetilde{W_1})) \vdash \mathsf{un}\ s_1(z) \,.\, (\boldsymbol{\nu} x_1 y_1) \ldots (\boldsymbol{\nu} x_t y_t)\overline{z}\langle\widetilde{x}\widetilde{y}\rangle \,.\, \mathbf{0}} \tag{30}$$

(29)

The tree is continued as follows:

$$\dfrac{\dfrac{\dfrac{z : !(\widetilde{V_1}\widetilde{W_1}) \vdash z : !(\widetilde{V_1}\widetilde{W_1}) \qquad \widetilde{x} : \widetilde{V_1}, \widetilde{y} : \widetilde{W_1} \vdash \widetilde{x} : \widetilde{V_1}, \widetilde{y} : \widetilde{W_1} \qquad \emptyset \vdash \mathbf{0}}{z : !(\widetilde{V_1}\widetilde{W_1}), x_1 : V_{1,1}, y_1 : \overline{V_{1,1}}, \ldots, x_t : V_{1,t}, y_t : \overline{V_{1,t}} \vdash \overline{z}\langle\widetilde{x}\widetilde{y}\rangle \,.\, \mathbf{0}}\,[T\text{-}Out]}{\vdots\ (t)}\,[T\text{-}Res]}{z : !(\widetilde{V_1}\widetilde{W_1}) \vdash (\boldsymbol{\nu} x_1 y_1) \ldots (\boldsymbol{\nu} x_t y_t)\overline{z}\langle\widetilde{x}\widetilde{y}\rangle \,.\, \mathbf{0}}\,[T\text{-}Res] \tag{30}$$

Each $W_{1,j} = \overline{V_{1,j}}$ by construction, so the subtree (29) is well-justified. All other name servers for $T_2, \ldots, T_q$ in (28) are typed in a similar way. The proof is thus concluded. ☺

7 Operational Correspondence

In this section we illustrate the relation between a process and its trio translation. From this, we define a weak bisimulation relation on processes, and conjecture that $(P, \mathcal{T}(P))$ is in this relation.

Example 11. Consider the well-typed process and its reductions in Figure 10: We have $U = !\mathsf{bool} \,.\, ?\mathsf{bool}$, and therefore $\mathcal{G}(U) = (!\mathsf{bool}, ?\mathsf{bool})$. We define the first name server as

$$N_U \triangleq \mathsf{un}\ s_U(z) \,.\, (\boldsymbol{\nu} w_1' z_1' : !\mathsf{bool})(\boldsymbol{\nu} w_2' z_2' : ?\mathsf{bool})\ \overline{z}\langle w_1' w_2' z_1' z_2'\rangle \,.\, \mathbf{0}$$

The bound names are chosen very conveniently, as will become clear later. Next, $T = !\mathsf{bool} \,.\, ?U$, and so, $\mathcal{G}(T) = (!\mathsf{bool}, ?\mathcal{G}(U)) = (!\mathsf{bool}, ?(!\mathsf{bool}, ?\mathsf{bool}))$. The second name

$$
\begin{array}{lcl}
P_1 & & (\boldsymbol{\nu} xy : T)(\overline{x}\langle \mathsf{true} \rangle \,.\, x(s) \,.\, \overline{s}\langle \mathsf{false} \rangle \,.\, s(a : \mathsf{bool}) \,.\, \mathbf{0} \\
 & & \quad \mid\; y(a : \mathsf{bool}) \,.\, (\boldsymbol{\nu} wz : U)\overline{y}\langle w \rangle \,.\, z(b : \mathsf{bool}) \,.\, \overline{z}\langle a \rangle \,.\, \mathbf{0}) \\
P_2 & \xrightarrow{\tau} & (\boldsymbol{\nu} xy : T')(\boldsymbol{\nu} wz : U)(x(s) \,.\, \overline{s}\langle \mathsf{false} \rangle \,.\, s(a : \mathsf{bool}) \,.\, \mathbf{0} \\
 & & \quad \mid\; \overline{y}\langle w \rangle \,.\, z(b : \mathsf{bool}) \,.\, \overline{z}\langle \mathsf{true} \rangle \,.\, \mathbf{0}) \\
P_3 & \xrightarrow{\tau} & (\boldsymbol{\nu} wz : U)(\overline{w}\langle \mathsf{false} \rangle \,.\, w(a : \mathsf{bool}) \,.\, \mathbf{0} \;\mid\; z(b : \mathsf{bool}) \,.\, \overline{z}\langle \mathsf{true} \rangle \,.\, \mathbf{0}) \\
P_4 & \xrightarrow{\tau} & (\boldsymbol{\nu} wz : U')(w(a : \mathsf{bool}) \,.\, \mathbf{0} \;\mid\; \overline{z}\langle \mathsf{true} \rangle \,.\, \mathbf{0}) \\
P_5 & \xrightarrow{\tau} & \mathbf{0}
\end{array}
$$

Figure 10: Example process reduction with delegation

server is therefore defined as follows:

$$N_T \triangleq \mathsf{un}\; s_T(z) \,.\, (\boldsymbol{\nu} x_1' y_1' : !\mathsf{bool})(\boldsymbol{\nu} x_2' y_2' : ?(!\mathsf{bool}, ?\mathsf{bool}))\overline{z}\langle x_1' x_2' y_1' y_2' \rangle \,.\, \mathbf{0}$$

The function will first create a trio for receiving names $\widetilde{x}$ and $\widetilde{y}$. After this, the composition rule will be applied and both parallel parts get a trio for input or output. Following that, one parallel process will get a trio that receives names $\widetilde{w}$ and $\widetilde{z}$, and both parallel processes will be broken down by input and output rules, ending in the rule for termination. We obtain the following translation of P in Figure 10:

$$
\begin{array}{ll}
 & \mathcal{T}(P_1) = (\boldsymbol{\nu} \widetilde{m}\widetilde{n})(\boldsymbol{\nu} s_U c_U)(\boldsymbol{\nu} s_T c_T)\; (\overline{m_1}\langle\rangle \,.\, \mathbf{0} \\
\text{(restr.)} & \mid\; n_1() \,.\, (\boldsymbol{\nu} rt)\overline{c_T}\langle t \rangle \,.\, r(x_1 x_2 y_1 y_2) \,.\, \overline{m_2}\langle x_1 x_2 y_1 y_2 \rangle \,.\, \mathbf{0} \\
\text{(parallel)} & \mid\; n_2(x_1 x_2 y_1 y_2) \,.\, \overline{m_3}\langle x_1 x_2 \rangle \,.\, \overline{m_8}\langle y_1 y_2 \rangle \,.\, \mathbf{0} \\
\text{(output)} & \mid\; n_3(x_1 x_2) \,.\, \overline{x_1}\langle \mathsf{true} \rangle \,.\, \overline{m_4}\langle x_2 \rangle \,.\, \mathbf{0} \\
\text{...} & \mid\; n_4(x_2) \,.\, x_2(s_1 s_2) \,.\, \overline{m_5}\langle s_1 s_2 \rangle \,.\, \mathbf{0} \\
 & \mid\; n_5(s_1 s_2) \,.\, \overline{s_1}\langle \mathsf{false} \rangle \,.\, \overline{m_6}\langle s_2 \rangle \,.\, \mathbf{0} \\
 & \mid\; n_6(s_2) \,.\, s_2(a_1) \,.\, \overline{m_7}\langle a_1 \rangle \,.\, \mathbf{0} \\
\text{(term.)} & \mid\; n_7(a_1) \,.\, \mathbf{0} \\
\text{(input)} & \mid\; n_8(y_1 y_2) \,.\, y_1(a_1) \,.\, \overline{m_9}\langle y_2 a_1 \rangle \,.\, \mathbf{0} \\
\text{(restr.)} & \mid\; n_9(y_2 a_1) \,.\, (\boldsymbol{\nu} rt)\overline{c_U}\langle t \rangle \,.\, r(w_1 w_2 z_1 z_2) \,.\, \overline{m_{10}}\langle y_2 a_1 w_1 w_2 z_1 z_2 \rangle \,.\, \mathbf{0} \\
\text{...} & \mid\; n_{10}(y_2 a_1 w_1 w_2 z_1 z_2) \,.\, \overline{y_2}\langle w_1 w_2 \rangle \,.\, \overline{m_{11}}\langle a_1 z_1 z_2 \rangle \,.\, \mathbf{0} \\
 & \mid\; n_{11}(a_1 z_1 z_2) \,.\, z_1(b_1) \,.\, \overline{m_{12}}\langle a_1 z_2 b_1 \rangle \,.\, \mathbf{0} \\
 & \mid\; n_{12}(a_1 z_2 b_1) \,.\, \overline{z_2}\langle a_1 \rangle \,.\, \overline{m_{13}}\langle a_1 b_1 \rangle \,.\, \mathbf{0} \\
\text{(term.)} & \mid\; n_{13}(a_1 b_1) \,.\, \mathbf{0} \\
\text{(name inventor)} & \mid\; \mathsf{un}\; s_U(z) \,.\, (\boldsymbol{\nu} w_1' z_1' : !\mathsf{bool})(\boldsymbol{\nu} w_2' z_2' : ?\mathsf{bool})\; \overline{z}\langle w_1' w_2' z_1' z_2' \rangle \,.\, \mathbf{0} \\
 & \mid\; \mathsf{un}\; s_T(z) \,.\, (\boldsymbol{\nu} x_1' y_1' : !\mathsf{bool})(\boldsymbol{\nu} x_2' y_2' : ?(!\mathsf{bool}, ?\mathsf{bool}))\overline{z}\langle x_1' x_2' y_1' y_2' \rangle \,.\, \mathbf{0})
\end{array}
$$

For the reduction chain, see Appendix B

7.1 Bisimulation

Now we wish to formulate an operational correspondence between a process P and its translation $\mathcal{T}(P)$. The concept of a weak or strong *open* bisimulation, as originally proposed in [8] is typically used for this. Strongly bisimilar processes can match any action and all subsequent actions, whereas weakly bisimilar processes are allowed to restructure themselves internally before and after matching an action. The latter relation may also be referred to as *observational equivalence*.

Because in our research we consider only closed processes, all actions are unobservable. Claiming that two well-typed processes in our typed calculus are weakly bisimilar in the traditional sense is not a very significant statement, because all of them eventually reach termination after a finite number of transitions. We will therefore introduce the notion of a *closed* weak bisimulation, making distinction between two different kinds of unobservable transitions. One is labeled a $\tau_{\mathcal{B}}$ transition, referring to a communication either between an m_k and n_k for some integer k or between an s_T and c_T for some type T. We then let τ_x refer to any other unobservable transition.

Definition 10 (*Closed weak bisimulation*)**.** A closed weak bisimulation is a binary relation $\mathcal{R}$ between processes, such that if $P\mathcal{R}Q$, then

(i) If $P \xrightarrow{\alpha} P'$, then

  (a) If $\alpha = \tau_x$, then $\exists Q'$ such that $Q \overset{\tau_{\mathcal{B}}}{\Longrightarrow} \xrightarrow{\tau_x} Q'$ and $P'\mathcal{R}Q'$,

  (b) If $\alpha = \tau_{\mathcal{B}}$, then $P'\mathcal{R}Q$.

(ii) The converse on the unobservable transitions emanating from Q, namely if $Q \xrightarrow{\alpha} Q'$, then

  (a) If $\alpha = \tau_x$, then $\exists P'$ such that $P \overset{\tau_{\mathcal{B}}}{\Longrightarrow} \xrightarrow{\tau_x} P'$ and $P'\mathcal{R}Q'$,

  (b) If $\alpha = \tau_{\mathcal{B}}$, then $P\mathcal{R}Q'$.

Two processes P and Q are called Jorge bisimilar, written $P \dot{\approx} Q$, iff there exists a closed weak bisimulation $\mathcal{R}$ such that $P\mathcal{R}Q$. The relation $\dot{\approx}$ is referred to as Jorge bisimilarity.

Lemma 5. The relation $\dot{\approx}$ is an equivalence relation.

Proof. See Appendix A. ☺

Example 12. Consider again the process in Figure 10 and the reduction of its translation (see Appendix B). The process P_1 is then Jorge bisimilar to $\mathcal{T}(P_1) = Q_1$ by the following relation.

$$\begin{aligned}\mathcal{R} = \{&(P_1, Q_1^1), (P_1, Q_1^2), (P_1, Q_1^3), (P_1, Q_1^4), (P_1, Q_1^5), (P_1, Q_1^6), (P_1, Q_1^7),\\ &(P_2, Q_2^1), (P_2, Q_2^{2a}), (P_2, Q_2^{2b}), (P_2, Q_2^3), (P_2, Q_2^4), (P_2, Q_2^5), (P_2, Q_2^6),\\ &(P_3, Q_3^1), (P_3, Q_3^{2a}), (P_3, Q_3^{2b}), (P_3, Q_3^3),\\ &(P_4, Q_4^1), (P_4, Q_4^{2a}), (P_4, Q_4^{2b}), (P_4, Q_4^3),\\ &(P_5, Q_5^1), (P_5, Q_5^{2a}), (P_5, Q_5^{2b}), (P_5, Q_5^3)\}\end{aligned}$$

The result that $P \dot{\approx} \mathcal{T}(P)$ follows from Definition 10 and a tedious examination of every pair of processes in $\mathcal{R}$. Every process in the relation has only one possible transition, except the processes Q_2^1, Q_3^1, Q_4^1, and Q_5^1. These processes are prepared to activate two trios for communication, and either trio can be activated first.

The processes P_3 and P_4 are both ready to do a single communication. For both processes, two internal steps of restructuring are done in the translation, one for each thread. Hence, P_3 is related to exactly three other processes, as is P_4. The process P_2 also does a single communication, but it first fetches new names from the name inventor in three internal steps. Therefore, P_2 is related to exactly six other processes. Process P_1 is similar to P_2, but is related to one extra process because of the parallel operator.

Finally, process P_5 is related to two extra processes, since there are still two trios for termination left, one for each thread. These two need to be added as a consequence of Property (iib) of Definition 10. Both processes P_5 and Q_5^3 cannot reduce further.

The results from Example 12 and [7] lead to the following conjecture:

Conjecture 1. Let P and $\mathcal{T}(P)$ be as in Definition 9. Then $P \dot{\approx} \mathcal{T}(P)$.

8 Conclusions

In this thesis, we have used the π-calculus to model concurrent communicating processes. A type system was introduced to be able to reason about safe and correct communication of processes in a formal way. After this, we introduced a function that breaks a process down into a trio translation, called a *concert*. The function has been designed to induce minimal types.

Our main result is that this breakdown function preserves typability. Since concerts obtain the full expressive power of the π-calculus (up to weak bisimulation), this is a major stepping stone to a very significant hypothesis: having two sequential forms, one for processes and one for types, might actually be redundant. If so, a programming language can be designed that does not have extremely long and complex protocols: every session would do exactly one thing.

The breakdown function maps a monadic syntax to a polyadic syntax. However, it is well-known that a polyadic process can always be encoded as a monadic one by first establishing a private session, and sending all names over this new channel. The type system did not include rules for polyadic communication, but the results will hold with a simple generalization of the rules.

An attempt has been made to prove closed weak bisimilarity between a process and its trio translation, inspired by the relation defined by Parrow [7]. This has turned out to be a notationally very heavy task, and was deemed outside the scope of this thesis. Examples strongly suggest, however, that the result will hold.

For future work, the breakdown function can be refined such that more information is available on how the names of the sequence $\widetilde{u}$ are typed in each trio. The construction of the list of types for each sequence is now done in the proof of Lemma 4, but can be integrated in the function

definition. This will also provide more insight into the procedure of transforming free names in the process to bound names in a trio.

We have assumed that processes did not contain replication. This is a disappointing restriction, as it prevents one to model infinite behaviour, found in many implemented systems. A forthcoming research is to apply the idea in this thesis to a language modeling higher-order session processes [4]. In this language, unbounded behaviour can be modeled without replication. Furthermore, weak bisimilarity relations are clearly defined for this typed language.

An acknowledgement of great gratitude goes out to Alan Arslanagic, for being beneficially critical of this work, and to Jorge Perez, who has been an inspiration to learn about session types, and to do research in general.

References

[1] H. P. Barendregt et al. *The Lambda Calculus: Its Syntax and Semantics*. North-Holland, 1984.

[2] K. Honda. Types for dyadic interaction. In *International Conference on Concurrency Theory*, pages 509–523. Springer, 1993.

[3] K. Honda, V. T. Vasconcelos, and M. Kubo. Language primitives and type discipline for structured communication-based programming. In *European Symposium on Programming*, pages 122–138. Springer, 1998.

[4] D. Kouzapas, J. A. Pérez, and N. Yoshida. On the relative expressiveness of higher-order session processes. In *European Symposium on Programming Languages and Systems*, pages 446–475. Springer, 2016.

[5] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, i. *Information and computation*, 100(1):1–40, 1992.

[6] D. Mostrous and V. T. Vasconcelos. Session typing for a featherweight Erlang. In *11th International Conference on Coordination Models and Languages*, volume 6721 of *LNCS*, pages 95–109. SPRINGER, 2011.

[7] J. Parrow. Trios in concert. In *Proof, Language, and Interaction, Essays in Honour of Robin Milner*, pages 623–638, 2000.

[8] D. Sangiorgi. A theory of bisimulation for the π-calculus. *Acta informatica*, 33(1):69–97, 1996.

[9] D. Sangiorgi and D. Walker. *The pi-calculus: a Theory of Mobile Processes*. Cambridge university press, 2003.

[10] K. Takeuchi, K. Honda, and M. Kubo. An interaction-based language and its typing system. In *International Conference on Parallel Architectures and Languages Europe*, pages 398–413. Springer, 1994.

[11] F. van Walree. Session types in cloud haskell. Master's thesis, Utrecht University, 2017.

[12] V. T. Vasconcelos. Fundamentals of session types. *Information and Computation*, 217:52 – 70, 2012.

[13] V. T. Vasconcelos, S. J. Gay, A. Ravara, N. Gesbert, and A. Z. Caldeira. Dynamic interfaces. In *International Workshop on Foundations of Object-Oriented Languages (FOOL)*, 2009.

Appendices

A Proof of Lemma 5

Lemma 3. The relation $\dot{\approx}$ as in Definition 10 is an equivalence relation.

Proof. We have to show reflexivity, symmetry, and transitivity of $\dot{\approx}$.

- Reflexivity is shown by providing a Jorge bisimulation $\mathcal{R}$ such that $P\mathcal{R}P$ for all processes P. This is achieved with the relation:
$$\mathcal{R} = \{(P, Q) : P \stackrel{\tau_{\mathcal{B}}}{\Rightarrow} Q \text{ or } Q \stackrel{\tau_{\mathcal{B}}}{\Rightarrow} P\}.$$
This relation now satisfies (i) and (ii) of Definition 10. The identity relation is indeed contained in $\mathcal{R}$, so $P\mathcal{R}P$ for all processes P.
- To show that $\dot{\approx}$ is symmetric, assume that $P \dot{\approx} Q$. Then there exists a Jorge bisimulation $\mathcal{R}$ such that $P\mathcal{R}Q$. Now define
$$\mathcal{S} = \{(Q, P) : P\mathcal{R}Q\},$$
Indeed, $(Q, P) \in \mathcal{S}$ if $P\mathcal{R}Q$. What remains is to show that (i) and (ii) hold for $\mathcal{S}$. For $(Q, P) \in \mathcal{S}$, we have $P\mathcal{R}Q$, and:
  (i) If $Q \stackrel{\alpha}{\to} Q'$, then
    (a) If $\alpha = \tau_x$, then because $P\mathcal{R}Q$, by property (ii.a), $\exists P'$, such that $P \stackrel{\tau_{\mathcal{B}}}{\Rightarrow}\stackrel{\tau_x}{\to} P'$ and $P'\mathcal{R}Q'$. Then by definition $(Q', P') \in \mathcal{S}$.
    (b) If $\alpha = \tau_{\mathcal{B}}$, then because $P\mathcal{R}Q$, by property (ii.b) also $P\mathcal{R}Q'$, and hence $(Q', P) \in \mathcal{S}$.
  (ii) The converse is completely analogous to (i), using property (i.a) and (i.b) of $\mathcal{R}$.
- Lastly, if $P \dot{\approx} Q$ by a Jorge bisimulation $\mathcal{R}$, and $Q \dot{\approx} R$ by $\mathcal{R}'$, we will show that $P \dot{\approx} R$ using the relation
$$\mathcal{S} = \mathcal{R}\mathcal{R}',$$
meaning that if $P\mathcal{S}R$, then there exists Q such that $P\mathcal{R}Q$ and $Q\mathcal{R}'R$, so indeed $(P, R) \in \mathcal{S}$. Furthermore,
  (i) If $P \stackrel{\alpha}{\to} P'$, then
    (a) If $\alpha = \tau_x$, then there exists Q' by property (i.a), such that $Q =: Q_0 \stackrel{\tau_{\mathcal{B}}}{\to} Q_1 \stackrel{\tau_{\mathcal{B}}}{\to} \cdots \stackrel{\tau_{\mathcal{B}}}{\to} Q_n \stackrel{\tau_x}{\to} Q'$ for some $n \geq 0$ and $P'\mathcal{R}Q'$. Furthermore, since $Q\mathcal{R}'R$, by (i.b) also $Q_n\mathcal{R}'R$. Then, if $Q_n \stackrel{\tau_x}{\to} Q'$, using property (i.a), there exists R' such that $R \stackrel{\tau_{\mathcal{B}}}{\Rightarrow}\stackrel{\tau_x}{\to} R'$ and $Q'\mathcal{R}'R'$. We have now found Q' and R' such that $R \stackrel{\tau_{\mathcal{B}}}{\Rightarrow}\stackrel{\tau_x}{\to} R'$, and $P'\mathcal{R}Q'$ and $Q'\mathcal{R}'R'$, so by definition $(P', R') \in \mathcal{S}$.
    (b) If $\alpha = \tau_{\mathcal{B}}$, we have $P'\mathcal{R}Q$ and also $Q\mathcal{R}'R$, so $P'\mathcal{S}R$.
  (ii) The converse is again completely analogous to (i).

☺

B Bisimulation Example

$$
\begin{aligned}
\mathcal{T}(P_1) = Q_1^1 = \quad & (\boldsymbol{\nu}\widetilde{m}\widetilde{n})(\boldsymbol{\nu}s_U c_U)(\boldsymbol{\nu}s_T c_T)\ (\overline{m_1}\langle\rangle \,.\, \mathbf{0} \\
& \mid n_1() \,.\, (\boldsymbol{\nu}rt)\overline{c_T}\langle t\rangle \,.\, r(x_1x_2y_1y_2) \,.\, \overline{m_2}\langle x_1x_2y_1y_2\rangle \,.\, \mathbf{0} \\
& \mid n_2(x_1x_2y_1y_2) \,.\, \overline{m_3}\langle x_1x_2\rangle \,.\, \overline{m_8}\langle y_1y_2\rangle \,.\, \mathbf{0} \\
& \mid n_3(x_1x_2) \,.\, \overline{x_1}\langle \mathsf{true}\rangle \,.\, \overline{m_4}\langle x_2\rangle \,.\, \mathbf{0} \\
& \mid n_4(x_2) \,.\, x_2(s_1s_2) \,.\, \overline{m_5}\langle s_1s_2\rangle \,.\, \mathbf{0} \\
& \mid n_5(s_1s_2) \,.\, \overline{s_1}\langle \mathsf{false}\rangle \,.\, \overline{m_6}\langle s_2\rangle \,.\, \mathbf{0} \\
& \mid n_6(s_2) \,.\, s_2(a_1) \,.\, \overline{m_7}\langle a_1\rangle \,.\, \mathbf{0} \\
& \mid n_7(a_1) \,.\, \mathbf{0} \\
& \mid n_8(y_1y_2) \,.\, y_1(a_1) \,.\, \overline{m_9}\langle y_2a_1\rangle \,.\, \mathbf{0} \\
& \mid n_9(y_2a_1) \,.\, (\boldsymbol{\nu}rt)\overline{c_U}\langle t\rangle \,.\, r(w_1w_2z_1z_2) \,.\, \overline{m_{10}}\langle y_2a_1w_1w_2z_1z_2\rangle \,.\, \mathbf{0} \\
& \mid n_{10}(y_2a_1w_1w_2z_1z_2) \,.\, \overline{y_2}\langle w_1w_2\rangle \,.\, \overline{m_{11}}\langle a_1z_1z_2\rangle \,.\, \mathbf{0} \\
& \mid n_{11}(a_1z_1z_2) \,.\, z_1(b_1) \,.\, \overline{m_{12}}\langle a_1z_2b_1\rangle \,.\, \mathbf{0} \\
& \mid n_{12}(a_1z_2b_1) \,.\, \overline{z_2}\langle a_1\rangle \,.\, \overline{m_{13}}\langle a_1b_1\rangle \,.\, \mathbf{0} \\
& \mid n_{13}(a_1b_1) \,.\, \mathbf{0} \\
& \mid \mathsf{un}\ s_U(z) \,.\, (\boldsymbol{\nu}w_1'z_1' :!\mathsf{bool})(\boldsymbol{\nu}w_2'z_2' :?\mathsf{bool})\ \overline{z}\langle w_1'w_2'z_1'z_2'\rangle \,.\, \mathbf{0} \\
& \mid \mathsf{un}\ s_T(z) \,.\, (\boldsymbol{\nu}x_1'y_1' :!\mathsf{bool})(\boldsymbol{\nu}x_2'y_2' :?(!\mathsf{bool}, ?\mathsf{bool}))\overline{z}\langle x_1'x_2'y_1'y_2'\rangle \,.\, \mathbf{0}
\end{aligned}
$$

$$
\begin{aligned}
\xrightarrow{\tau_{\mathcal{B}}} \quad & Q_1^2 = (\boldsymbol{\nu}\widetilde{m}\widetilde{n})(\boldsymbol{\nu}s_U c_U)(\boldsymbol{\nu}s_T c_T) \\
(\quad & (\boldsymbol{\nu}rt)\overline{c_T}\langle t\rangle \,.\, r(x_1x_2y_1y_2) \,.\, \overline{m_2}\langle x_1x_2y_1y_2\rangle \,.\, \mathbf{0} \\
& \mid n_2(x_1x_2y_1y_2) \,.\, \overline{m_3}\langle x_1x_2\rangle \,.\, \overline{m_8}\langle y_1y_2\rangle \,.\, \mathbf{0} \\
& \mid n_3(x_1x_2) \,.\, \overline{x_1}\langle \mathsf{true}\rangle \,.\, \overline{m_4}\langle x_2\rangle \,.\, \mathbf{0} \\
& \mid n_4(x_2) \,.\, x_2(s_1s_2) \,.\, \overline{m_5}\langle s_1s_2\rangle \,.\, \mathbf{0} \\
& \mid n_5(s_1s_2) \,.\, \overline{s_1}\langle \mathsf{false}\rangle \,.\, \overline{m_6}\langle s_2\rangle \,.\, \mathbf{0} \\
& \mid n_6(s_2) \,.\, s_2(a_1) \,.\, \overline{m_7}\langle a_1\rangle \,.\, \mathbf{0} \\
& \mid n_7(a_1) \,.\, \mathbf{0} \\
& \mid n_8(y_1y_2) \,.\, y_1(a_1) \,.\, \overline{m_9}\langle y_2a_1\rangle \,.\, \mathbf{0} \\
& \mid n_9(y_2a_1) \,.\, (\boldsymbol{\nu}rt)\overline{c_U}\langle t\rangle \,.\, r(w_1w_2z_1z_2) \,.\, \overline{m_{10}}\langle y_2a_1w_1w_2z_1z_2\rangle \,.\, \mathbf{0} \\
& \mid n_{10}(y_2a_1w_1w_2z_1z_2) \,.\, \overline{y_2}\langle w_1w_2\rangle \,.\, \overline{m_{11}}\langle a_1z_1z_2\rangle \,.\, \mathbf{0} \\
& \mid n_{11}(a_1z_1z_2) \,.\, z_1(b_1) \,.\, \overline{m_{12}}\langle a_1z_2b_1\rangle \,.\, \mathbf{0} \\
& \mid n_{12}(a_1z_2b_1) \,.\, \overline{z_2}\langle a_1\rangle \,.\, \overline{m_{13}}\langle a_1b_1\rangle \,.\, \mathbf{0} \\
& \mid n_{13}(a_1b_1) \,.\, \mathbf{0} \\
& \mid \mathsf{un}\ s_U(z) \,.\, (\boldsymbol{\nu}w_1'z_1' :!\mathsf{bool})(\boldsymbol{\nu}w_2'z_2' :?\mathsf{bool})\ \overline{z}\langle w_1'w_2'z_1'z_2'\rangle \,.\, \mathbf{0} \\
& \mid \mathsf{un}\ s_T(z) \,.\, (\boldsymbol{\nu}x_1'y_1' :!\mathsf{bool})(\boldsymbol{\nu}x_2'y_2' :?(!\mathsf{bool}, ?\mathsf{bool}))\overline{z}\langle x_1'x_2'y_1'y_2'\rangle \,.\, \mathbf{0}
\end{aligned}
$$

$\xrightarrow{\tau_{\mathcal{B}}}$ $Q_1^3 = (\boldsymbol{\nu}\widetilde{m}\widetilde{n})(\boldsymbol{\nu}s_U c_U)(\boldsymbol{\nu}s_T c_T)$

($(\boldsymbol{\nu}rt)(r(x_1x_2y_1y_2) \,.\, \overline{m_2}\langle x_1x_2y_1y_2\rangle \,.\, \mathbf{0}$

| $(\boldsymbol{\nu}x_1'y_1' :!\mathsf{bool})(\boldsymbol{\nu}x_2'y_2' :?(!\mathsf{bool}, ?\mathsf{bool}))\bar{t}\langle x_1'x_2'y_1'y_2'\rangle \,.\, \mathbf{0})$

| $n_2(x_1x_2y_1y_2) \,.\, \overline{m_3}\langle x_1x_2\rangle \,.\, \overline{m_8}\langle y_1y_2\rangle \,.\, \mathbf{0}$

| $n_3(x_1x_2) \,.\, \overline{x_1}\langle \mathsf{true}\rangle \,.\, \overline{m_4}\langle x_2\rangle \,.\, \mathbf{0}$

| $n_4(x_2) \,.\, x_2(s_1s_2) \,.\, \overline{m_5}\langle s_1s_2\rangle \,.\, \mathbf{0}$

| $n_5(s_1s_2) \,.\, \overline{s_1}\langle \mathsf{false}\rangle \,.\, \overline{m_6}\langle s_2\rangle \,.\, \mathbf{0}$

| $n_6(s_2) \,.\, s_2(a_1) \,.\, \overline{m_7}\langle a_1\rangle \,.\, \mathbf{0}$

| $n_7(a_1) \,.\, \mathbf{0}$

| $n_8(y_1y_2) \,.\, y_1(a_1) \,.\, \overline{m_9}\langle y_2a_1\rangle \,.\, \mathbf{0}$

| $n_9(y_2a_1) \,.\, (\boldsymbol{\nu}rt)\overline{c_U}\langle t\rangle \,.\, r(w_1w_2z_1z_2) \,.\, \overline{m_{10}}\langle y_2a_1w_1w_2z_1z_2\rangle \,.\, \mathbf{0}$

| $n_{10}(y_2a_1w_1w_2z_1z_2) \,.\, \overline{y_2}\langle w_1w_2\rangle \,.\, \overline{m_{11}}\langle a_1z_1z_2\rangle \,.\, \mathbf{0}$

| $n_{11}(a_1z_1z_2) \,.\, z_1(b_1) \,.\, \overline{m_{12}}\langle a_1z_2b_1\rangle \,.\, \mathbf{0}$

| $n_{12}(a_1z_2b_1) \,.\, \overline{z_2}\langle a_1\rangle \,.\, \overline{m_{13}}\langle a_1b_1\rangle \,.\, \mathbf{0}$

| $n_{13}(a_1b_1) \,.\, \mathbf{0}$

| $\mathsf{un}\ s_U(z) \,.\, (\boldsymbol{\nu}w_1'z_1' :!\mathsf{bool})(\boldsymbol{\nu}w_2'z_2' :?\mathsf{bool})\ \overline{z}\langle w_1'w_2'z_1'z_2'\rangle \,.\, \mathbf{0}$

| $\mathsf{un}\ s_T(z) \,.\, (\boldsymbol{\nu}x_1'y_1' :!\mathsf{bool})(\boldsymbol{\nu}x_2'y_2' :?(!\mathsf{bool}, ?\mathsf{bool}))\overline{z}\langle x_1'x_2'y_1'y_2'\rangle \,.\, \mathbf{0}$

$\xrightarrow{\tau_{\mathcal{B}}}$ $Q_1^4 = (\boldsymbol{\nu}\widetilde{m}\widetilde{n})(\boldsymbol{\nu}s_U c_U)(\boldsymbol{\nu}s_T c_T)$

($(\boldsymbol{\nu}x_1'y_1' :!\mathsf{bool})(\boldsymbol{\nu}x_2'y_2' :?(!\mathsf{bool}, ?\mathsf{bool}))\overline{m_2}\langle x_1'x_2'y_1'y_2'\rangle \,.\, \mathbf{0}$

| $n_2(x_1x_2y_1y_2) \,.\, \overline{m_3}\langle x_1x_2\rangle \,.\, \overline{m_8}\langle y_1y_2\rangle \,.\, \mathbf{0}$

| $n_3(x_1x_2) \,.\, \overline{x_1}\langle \mathsf{true}\rangle \,.\, \overline{m_4}\langle x_2\rangle \,.\, \mathbf{0}$

| $n_4(x_2) \,.\, x_2(s_1s_2) \,.\, \overline{m_5}\langle s_1s_2\rangle \,.\, \mathbf{0}$

| $n_5(s_1s_2) \,.\, \overline{s_1}\langle \mathsf{false}\rangle \,.\, \overline{m_6}\langle s_2\rangle \,.\, \mathbf{0}$

| $n_6(s_2) \,.\, s_2(a_1) \,.\, \overline{m_7}\langle a_1\rangle \,.\, \mathbf{0}$

| $n_7(a_1) \,.\, \mathbf{0}$

| $n_8(y_1y_2) \,.\, y_1(a_1) \,.\, \overline{m_9}\langle y_2a_1\rangle \,.\, \mathbf{0}$

| $n_9(y_2a_1) \,.\, (\boldsymbol{\nu}rt)\overline{c_U}\langle t\rangle \,.\, r(w_1w_2z_1z_2) \,.\, \overline{m_{10}}\langle y_2a_1w_1w_2z_1z_2\rangle \,.\, \mathbf{0}$

| $n_{10}(y_2a_1w_1w_2z_1z_2) \,.\, \overline{y_2}\langle w_1w_2\rangle \,.\, \overline{m_{11}}\langle a_1z_1z_2\rangle \,.\, \mathbf{0}$

| $n_{11}(a_1z_1z_2) \,.\, z_1(b_1) \,.\, \overline{m_{12}}\langle a_1z_2b_1\rangle \,.\, \mathbf{0}$

| $n_{12}(a_1z_2b_1) \,.\, \overline{z_2}\langle a_1\rangle \,.\, \overline{m_{13}}\langle a_1b_1\rangle \,.\, \mathbf{0}$

| $n_{13}(a_1b_1) \,.\, \mathbf{0}$

| $\mathsf{un}\ s_U(z) \,.\, (\boldsymbol{\nu}w_1'z_1' :!\mathsf{bool})(\boldsymbol{\nu}w_2'z_2' :?\mathsf{bool})\ \overline{z}\langle w_1'w_2'z_1'z_2'\rangle \,.\, \mathbf{0}$

| $\mathsf{un}\ s_T(z) \,.\, (\boldsymbol{\nu}x_1'y_1' :!\mathsf{bool})(\boldsymbol{\nu}x_2'y_2' :?(!\mathsf{bool}, ?\mathsf{bool}))\overline{z}\langle x_1'x_2'y_1'y_2'\rangle \,.\, \mathbf{0}$

$$
\begin{array}{ll}
\xrightarrow{\tau_{\mathcal{B}}} & Q_1^5 = (\boldsymbol{\nu}\widetilde{m}\widetilde{n})(\boldsymbol{\nu}s_U c_U)(\boldsymbol{\nu}s_T c_T) \\
(& (\boldsymbol{\nu}x_1'y_1' :!\mathsf{bool})(\boldsymbol{\nu}x_2'y_2' :?(!\mathsf{bool}, ?\mathsf{bool}))\overline{m_3}\langle x_1'x_2'\rangle \,.\, \overline{m_8}\langle y_1'y_2'\rangle \,.\, \mathbf{0} \\
& \mid\ n_3(x_1x_2) \,.\, \overline{x_1}\langle \mathsf{true}\rangle \,.\, \overline{m_4}\langle x_2\rangle \,.\, \mathbf{0} \\
& \mid\ n_4(x_2) \,.\, x_2(s_1s_2) \,.\, \overline{m_5}\langle s_1s_2\rangle \,.\, \mathbf{0} \\
& \mid\ n_5(s_1s_2) \,.\, \overline{s_1}\langle \mathsf{false}\rangle \,.\, \overline{m_6}\langle s_2\rangle \,.\, \mathbf{0} \\
& \mid\ n_6(s_2) \,.\, s_2(a_1) \,.\, \overline{m_7}\langle a_1\rangle \,.\, \mathbf{0} \\
& \mid\ n_7(a_1) \,.\, \mathbf{0} \\
& \mid\ n_8(y_1y_2) \,.\, y_1(a_1) \,.\, \overline{m_9}\langle y_2a_1\rangle \,.\, \mathbf{0} \\
& \mid\ n_9(y_2a_1) \,.\, (\boldsymbol{\nu}rt)\overline{c_U}\langle t\rangle \,.\, r(w_1w_2z_1z_2) \,.\, \overline{m_{10}}\langle y_2a_1w_1w_2z_1z_2\rangle \,.\, \mathbf{0} \\
& \mid\ n_{10}(y_2a_1w_1w_2z_1z_2) \,.\, \overline{y_2}\langle w_1w_2\rangle \,.\, \overline{m_{11}}\langle a_1z_1z_2\rangle \,.\, \mathbf{0} \\
& \mid\ n_{11}(a_1z_1z_2) \,.\, z_1(b_1) \,.\, \overline{m_{12}}\langle a_1z_2b_1\rangle \,.\, \mathbf{0} \\
& \mid\ n_{12}(a_1z_2b_1) \,.\, \overline{z_2}\langle a_1\rangle \,.\, \overline{m_{13}}\langle a_1b_1\rangle \,.\, \mathbf{0} \\
& \mid\ n_{13}(a_1b_1) \,.\, \mathbf{0} \\
& \mid\ \mathsf{un}\ s_U(z) \,.\, (\boldsymbol{\nu}w_1'z_1' :!\mathsf{bool})(\boldsymbol{\nu}w_2'z_2' :?\mathsf{bool})\ \overline{z}\langle w_1'w_2'z_1'z_2'\rangle \,.\, \mathbf{0} \\
& \mid\ \mathsf{un}\ s_T(z) \,.\, (\boldsymbol{\nu}x_1'y_1' :!\mathsf{bool})(\boldsymbol{\nu}x_2'y_2' :?(!\mathsf{bool}, ?\mathsf{bool}))\overline{z}\langle x_1'x_2'y_1'y_2'\rangle \,.\, \mathbf{0}
\end{array}
$$

$$
\begin{array}{ll}
\xrightarrow{\tau_{\mathcal{B}}}\ Q_1^6 \xrightarrow{\tau_{\mathcal{B}}} & Q_1^7 = (\boldsymbol{\nu}\widetilde{m}\widetilde{n})(\boldsymbol{\nu}s_U c_U)(\boldsymbol{\nu}s_T c_T) \\
(& (\boldsymbol{\nu}x_1'y_1' :!\mathsf{bool})(\boldsymbol{\nu}x_2'y_2' :?(!\mathsf{bool}, ?\mathsf{bool})) \\
& (\overline{x_1'}\langle \mathsf{true}\rangle \,.\, \overline{m_4}\langle x_2'\rangle \,.\, \mathbf{0} \ \mid\ y_1'(a_1) \,.\, \overline{m_9}\langle y_2'a_1\rangle \,.\, \mathbf{0}) \\
& \mid\ n_4(x_2) \,.\, x_2(s_1s_2) \,.\, \overline{m_5}\langle s_1s_2\rangle \,.\, \mathbf{0} \\
& \mid\ n_5(s_1s_2) \,.\, \overline{s_1}\langle \mathsf{false}\rangle \,.\, \overline{m_6}\langle s_2\rangle \,.\, \mathbf{0} \\
& \mid\ n_6(s_2) \,.\, s_2(a_1) \,.\, \overline{m_7}\langle a_1\rangle \,.\, \mathbf{0} \\
& \mid\ n_7(a_1) \,.\, \mathbf{0} \\
& \mid\ n_9(y_2a_1) \,.\, (\boldsymbol{\nu}rt)\overline{c_U}\langle t\rangle \,.\, r(w_1w_2z_1z_2) \,.\, \overline{m_{10}}\langle y_2a_1w_1w_2z_1z_2\rangle \,.\, \mathbf{0} \\
& \mid\ n_{10}(y_2a_1w_1w_2z_1z_2) \,.\, \overline{y_2}\langle w_1w_2\rangle \,.\, \overline{m_{11}}\langle a_1z_1z_2\rangle \,.\, \mathbf{0} \\
& \mid\ n_{11}(a_1z_1z_2) \,.\, z_1(b_1) \,.\, \overline{m_{12}}\langle a_1z_2b_1\rangle \,.\, \mathbf{0} \\
& \mid\ n_{12}(a_1z_2b_1) \,.\, \overline{z_2}\langle a_1\rangle \,.\, \overline{m_{13}}\langle a_1b_1\rangle \,.\, \mathbf{0} \\
& \mid\ n_{13}(a_1b_1) \,.\, \mathbf{0} \\
& \mid\ \mathsf{un}\ s_U(z) \,.\, (\boldsymbol{\nu}w_1'z_1' :!\mathsf{bool})(\boldsymbol{\nu}w_2'z_2' :?\mathsf{bool})\ \overline{z}\langle w_1'w_2'z_1'z_2'\rangle \,.\, \mathbf{0} \\
& \mid\ \mathsf{un}\ s_T(z) \,.\, (\boldsymbol{\nu}x_1'y_1' :!\mathsf{bool})(\boldsymbol{\nu}x_2'y_2' :?(!\mathsf{bool}, ?\mathsf{bool}))\overline{z}\langle x_1'x_2'y_1'y_2'\rangle \,.\, \mathbf{0}
\end{array}
$$

$$
\begin{array}{ll}
\xrightarrow{\tau_x} & Q_2^1 = (\boldsymbol{\nu}\widetilde{m}\widetilde{n})(\boldsymbol{\nu}s_U c_U)(\boldsymbol{\nu}s_T c_T) \\
(& (\boldsymbol{\nu}x_2'y_2' :?(!\mathsf{bool}, ?\mathsf{bool}))(\overline{m_4}\langle x_2'\rangle \,.\, \mathbf{0} \ \mid\ \overline{m_9}\langle y_2'\mathsf{true}\rangle \,.\, \mathbf{0}) \\
& \mid\ n_4(x_2) \,.\, x_2(s_1s_2) \,.\, \overline{m_5}\langle s_1s_2\rangle \,.\, \mathbf{0} \\
& \mid\ n_5(s_1s_2) \,.\, \overline{s_1}\langle \mathsf{false}\rangle \,.\, \overline{m_6}\langle s_2\rangle \,.\, \mathbf{0} \\
& \mid\ n_6(s_2) \,.\, s_2(a_1) \,.\, \overline{m_7}\langle a_1\rangle \,.\, \mathbf{0} \\
& \mid\ n_7(a_1) \,.\, \mathbf{0} \\
& \mid\ n_9(y_2a_1) \,.\, (\boldsymbol{\nu}rt)\overline{c_U}\langle t\rangle \,.\, r(w_1w_2z_1z_2) \,.\, \overline{m_{10}}\langle y_2a_1w_1w_2z_1z_2\rangle \,.\, \mathbf{0} \\
& \mid\ n_{10}(y_2a_1w_1w_2z_1z_2) \,.\, \overline{y_2}\langle w_1w_2\rangle \,.\, \overline{m_{11}}\langle a_1z_1z_2\rangle \,.\, \mathbf{0} \\
& \mid\ n_{11}(a_1z_1z_2) \,.\, z_1(b_1) \,.\, \overline{m_{12}}\langle a_1z_2b_1\rangle \,.\, \mathbf{0} \\
& \mid\ n_{12}(a_1z_2b_1) \,.\, \overline{z_2}\langle a_1\rangle \,.\, \overline{m_{13}}\langle a_1b_1\rangle \,.\, \mathbf{0} \\
& \mid\ n_{13}(a_1b_1) \,.\, \mathbf{0} \\
& \mid\ \mathsf{un}\ s_U(z) \,.\, (\boldsymbol{\nu}w_1'z_1' :!\mathsf{bool})(\boldsymbol{\nu}w_2'z_2' :?\mathsf{bool})\ \overline{z}\langle w_1'w_2'z_1'z_2'\rangle \,.\, \mathbf{0} \\
& \mid\ \mathsf{un}\ s_T(z) \,.\, (\boldsymbol{\nu}x_1'y_1' :!\mathsf{bool})(\boldsymbol{\nu}x_2'y_2' :?(!\mathsf{bool}, ?\mathsf{bool}))\overline{z}\langle x_1'x_2'y_1'y_2'\rangle \,.\, \mathbf{0}
\end{array}
$$

$\xrightarrow{\tau_{\mathcal{B}}} Q_2^2 \xrightarrow{\tau_{\mathcal{B}}} \quad Q_2^3 = (\boldsymbol{\nu}\widetilde{m}\widetilde{n})(\boldsymbol{\nu}s_U c_U)(\boldsymbol{\nu}s_T c_T)$

$(\quad (\boldsymbol{\nu}x_2'y_2' :?(!\mathsf{bool}, ?\mathsf{bool}))(x_2'(s_1s_2) \,.\, \overline{m_5}\langle s_1s_2\rangle \,.\, \mathbf{0}$

$\quad\quad \mid\ (\boldsymbol{\nu}rt)\overline{c_U}\langle t\rangle \,.\, r(w_1w_2z_1z_2) \,.\, \overline{m_{10}}\langle y_2'\mathsf{true}w_1w_2z_1z_2\rangle \,.\, \mathbf{0})$

$\quad\mid\ n_5(s_1s_2) \,.\, \overline{s_1}\langle\mathsf{false}\rangle \,.\, \overline{m_6}\langle s_2\rangle \,.\, \mathbf{0}$

$\quad\mid\ n_6(s_2) \,.\, s_2(a_1) \,.\, \overline{m_7}\langle a_1\rangle \,.\, \mathbf{0}$

$\quad\mid\ n_7(a_1) \,.\, \mathbf{0}$

$\quad\mid\ n_{10}(y_2a_1w_1w_2z_1z_2) \,.\, \overline{y_2}\langle w_1w_2\rangle \,.\, \overline{m_{11}}\langle a_1z_1z_2\rangle \,.\, \mathbf{0}$

$\quad\mid\ n_{11}(a_1z_1z_2) \,.\, z_1(b_1) \,.\, \overline{m_{12}}\langle a_1z_2b_1\rangle \,.\, \mathbf{0}$

$\quad\mid\ n_{12}(a_1z_2b_1) \,.\, \overline{z_2}\langle a_1\rangle \,.\, \overline{m_{13}}\langle a_1b_1\rangle \,.\, \mathbf{0}$

$\quad\mid\ n_{13}(a_1b_1) \,.\, \mathbf{0}$

$\quad\mid\ \mathsf{un}\ s_U(z) \,.\, (\boldsymbol{\nu}w_1'z_1' :!\mathsf{bool})(\boldsymbol{\nu}w_2'z_2' :?\mathsf{bool})\ \overline{z}\langle w_1'w_2'z_1'z_2'\rangle \,.\, \mathbf{0}$

$\quad\mid\ \mathsf{un}\ s_T(z) \,.\, (\boldsymbol{\nu}x_1'y_1' :!\mathsf{bool})(\boldsymbol{\nu}x_2'y_2' :?(!\mathsf{bool}, ?\mathsf{bool}))\overline{z}\langle x_1'x_2'y_1'y_2'\rangle \,.\, \mathbf{0}$

$\xrightarrow{\tau_{\mathcal{B}}} \quad Q_2^4 = (\boldsymbol{\nu}\widetilde{m}\widetilde{n})(\boldsymbol{\nu}s_U c_U)(\boldsymbol{\nu}s_T c_T)$

$(\quad (\boldsymbol{\nu}x_2'y_2' :?(!\mathsf{bool}, ?\mathsf{bool}))(x_2'(s_1s_2) \,.\, \overline{m_5}\langle s_1s_2\rangle \,.\, \mathbf{0}$

$\quad\quad \mid\ (\boldsymbol{\nu}rt)(r(w_1w_2z_1z_2) \,.\, \overline{m_{10}}\langle y_2'\mathsf{true}w_1w_2z_1z_2\rangle \,.\, \mathbf{0}$

$\quad\quad\quad \mid\ (\boldsymbol{\nu}w_1'z_1' :!\mathsf{bool})(\boldsymbol{\nu}w_2'z_2' :?\mathsf{bool})\ \overline{t}\langle w_1'w_2'z_1'z_2'\rangle \,.\, \mathbf{0}))$

$\quad\mid\ n_5(s_1s_2) \,.\, \overline{s_1}\langle\mathsf{false}\rangle \,.\, \overline{m_6}\langle s_2\rangle \,.\, \mathbf{0}$

$\quad\mid\ n_6(s_2) \,.\, s_2(a_1) \,.\, \overline{m_7}\langle a_1\rangle \,.\, \mathbf{0}$

$\quad\mid\ n_7(a_1) \,.\, \mathbf{0}$

$\quad\mid\ n_{10}(y_2a_1w_1w_2z_1z_2) \,.\, \overline{y_2}\langle w_1w_2\rangle \,.\, \overline{m_{11}}\langle a_1z_1z_2\rangle \,.\, \mathbf{0}$

$\quad\mid\ n_{11}(a_1z_1z_2) \,.\, z_1(b_1) \,.\, \overline{m_{12}}\langle a_1z_2b_1\rangle \,.\, \mathbf{0}$

$\quad\mid\ n_{12}(a_1z_2b_1) \,.\, \overline{z_2}\langle a_1\rangle \,.\, \overline{m_{13}}\langle a_1b_1\rangle \,.\, \mathbf{0}$

$\quad\mid\ n_{13}(a_1b_1) \,.\, \mathbf{0}$

$\quad\mid\ \mathsf{un}\ s_U(z) \,.\, (\boldsymbol{\nu}w_1'z_1' :!\mathsf{bool})(\boldsymbol{\nu}w_2'z_2' :?\mathsf{bool})\ \overline{z}\langle w_1'w_2'z_1'z_2'\rangle \,.\, \mathbf{0}$

$\quad\mid\ \mathsf{un}\ s_T(z) \,.\, (\boldsymbol{\nu}x_1'y_1' :!\mathsf{bool})(\boldsymbol{\nu}x_2'y_2' :?(!\mathsf{bool}, ?\mathsf{bool}))\overline{z}\langle x_1'x_2'y_1'y_2'\rangle \,.\, \mathbf{0}$

$\xrightarrow{\tau_{\mathcal{B}}} \quad Q_2^5 = (\boldsymbol{\nu}\widetilde{m}\widetilde{n})(\boldsymbol{\nu}s_U c_U)(\boldsymbol{\nu}s_T c_T)$

$(\quad (\boldsymbol{\nu}x_2'y_2' :?(!\mathsf{bool}, ?\mathsf{bool}))(\boldsymbol{\nu}w_1'z_1' :!\mathsf{bool})(\boldsymbol{\nu}w_2'z_2' :?\mathsf{bool})$

$\quad (x_2'(s_1s_2) \,.\, \overline{m_5}\langle s_1s_2\rangle \,.\, \mathbf{0}\ \mid\ \overline{m_{10}}\langle y_2'\mathsf{true}w_1'w_2'z_1'z_2'\rangle \,.\, \mathbf{0})$

$\quad\mid\ n_5(s_1s_2) \,.\, \overline{s_1}\langle\mathsf{false}\rangle \,.\, \overline{m_6}\langle s_2\rangle \,.\, \mathbf{0}$

$\quad\mid\ n_6(s_2) \,.\, s_2(a_1) \,.\, \overline{m_7}\langle a_1\rangle \,.\, \mathbf{0}$

$\quad\mid\ n_7(a_1) \,.\, \mathbf{0}$

$\quad\mid\ n_{10}(y_2a_1w_1w_2z_1z_2) \,.\, \overline{y_2}\langle w_1w_2\rangle \,.\, \overline{m_{11}}\langle a_1z_1z_2\rangle \,.\, \mathbf{0}$

$\quad\mid\ n_{11}(a_1z_1z_2) \,.\, z_1(b_1) \,.\, \overline{m_{12}}\langle a_1z_2b_1\rangle \,.\, \mathbf{0}$

$\quad\mid\ n_{12}(a_1z_2b_1) \,.\, \overline{z_2}\langle a_1\rangle \,.\, \overline{m_{13}}\langle a_1b_1\rangle \,.\, \mathbf{0}$

$\quad\mid\ n_{13}(a_1b_1) \,.\, \mathbf{0}$

$\quad\mid\ \mathsf{un}\ s_U(z) \,.\, (\boldsymbol{\nu}w_1'z_1' :!\mathsf{bool})(\boldsymbol{\nu}w_2'z_2' :?\mathsf{bool})\ \overline{z}\langle w_1'w_2'z_1'z_2'\rangle \,.\, \mathbf{0}$

$\quad\mid\ \mathsf{un}\ s_T(z) \,.\, (\boldsymbol{\nu}x_1'y_1' :!\mathsf{bool})(\boldsymbol{\nu}x_2'y_2' :?(!\mathsf{bool}, ?\mathsf{bool}))\overline{z}\langle x_1'x_2'y_1'y_2'\rangle \,.\, \mathbf{0}$

$$
\begin{array}{ll}
\xrightarrow{\tau_{\mathcal{B}}} & Q_2^6 = (\boldsymbol{\nu}\widetilde{m}\widetilde{n})(\boldsymbol{\nu}s_U c_U)(\boldsymbol{\nu}s_T c_T) \\
(& (\boldsymbol{\nu}x_2'y_2' :?(!\mathsf{bool}, ?\mathsf{bool}))(\boldsymbol{\nu}w_1'z_1' :!\mathsf{bool})(\boldsymbol{\nu}w_2'z_2' :?\mathsf{bool}) \\
& (x_2'(s_1s_2) \,.\, \overline{m_5}\langle s_1s_2\rangle \,.\, \mathbf{0} \mid \overline{y_2'}\langle w_1'w_2'\rangle \,.\, \overline{m_{11}}\langle \mathsf{true} z_1'z_2'\rangle \,.\, \mathbf{0}) \\
& \mid n_5(s_1s_2) \,.\, \overline{s_1}\langle \mathsf{false}\rangle \,.\, \overline{m_6}\langle s_2\rangle \,.\, \mathbf{0} \\
& \mid n_6(s_2) \,.\, s_2(a_1) \,.\, \overline{m_7}\langle a_1\rangle \,.\, \mathbf{0} \\
& \mid n_7(a_1) \,.\, \mathbf{0} \\
& \mid n_{11}(a_1z_1z_2) \,.\, z_1(b_1) \,.\, \overline{m_{12}}\langle a_1z_2b_1\rangle \,.\, \mathbf{0} \\
& \mid n_{12}(a_1z_2b_1) \,.\, \overline{z_2}\langle a_1\rangle \,.\, \overline{m_{13}}\langle a_1b_1\rangle \,.\, \mathbf{0} \\
& \mid n_{13}(a_1b_1) \,.\, \mathbf{0} \\
& \mid \mathsf{un}\ s_U(z) \,.\, (\boldsymbol{\nu}w_1'z_1' :!\mathsf{bool})(\boldsymbol{\nu}w_2'z_2' :?\mathsf{bool})\ \overline{z}\langle w_1'w_2'z_1'z_2'\rangle \,.\, \mathbf{0} \\
& \mid \mathsf{un}\ s_T(z) \,.\, (\boldsymbol{\nu}x_1'y_1' :!\mathsf{bool})(\boldsymbol{\nu}x_2'y_2' :?(!\mathsf{bool}, ?\mathsf{bool}))\overline{z}\langle x_1'x_2'y_1'y_2'\rangle \,.\, \mathbf{0}
\end{array}
$$

$$
\begin{array}{ll}
\xrightarrow{\tau_{x_1}} & Q_3^1 = (\boldsymbol{\nu}\widetilde{m}\widetilde{n})(\boldsymbol{\nu}s_U c_U)(\boldsymbol{\nu}s_T c_T) \\
(& (\boldsymbol{\nu}w_1'z_1' :!\mathsf{bool})(\boldsymbol{\nu}w_2'z_2' :?\mathsf{bool}) \\
& (\overline{m_5}\langle w_1'w_2'\rangle \,.\, \mathbf{0} \mid \overline{m_{11}}\langle \mathsf{true} z_1'z_2'\rangle \,.\, \mathbf{0}) \\
& \mid n_5(s_1s_2) \,.\, \overline{s_1}\langle \mathsf{false}\rangle \,.\, \overline{m_6}\langle s_2\rangle \,.\, \mathbf{0} \\
& \mid n_6(s_2) \,.\, s_2(a_1) \,.\, \overline{m_7}\langle a_1\rangle \,.\, \mathbf{0} \\
& \mid n_7(a_1) \,.\, \mathbf{0} \\
& \mid n_{11}(a_1z_1z_2) \,.\, z_1(b_1) \,.\, \overline{m_{12}}\langle a_1z_2b_1\rangle \,.\, \mathbf{0} \\
& \mid n_{12}(a_1z_2b_1) \,.\, \overline{z_2}\langle a_1\rangle \,.\, \overline{m_{13}}\langle a_1b_1\rangle \,.\, \mathbf{0} \\
& \mid n_{13}(a_1b_1) \,.\, \mathbf{0} \\
& \mid \mathsf{un}\ s_U(z) \,.\, (\boldsymbol{\nu}w_1'z_1' :!\mathsf{bool})(\boldsymbol{\nu}w_2'z_2' :?\mathsf{bool})\ \overline{z}\langle w_1'w_2'z_1'z_2'\rangle \,.\, \mathbf{0} \\
& \mid \mathsf{un}\ s_T(z) \,.\, (\boldsymbol{\nu}x_1'y_1' :!\mathsf{bool})(\boldsymbol{\nu}x_2'y_2' :?(!\mathsf{bool}, ?\mathsf{bool}))\overline{z}\langle x_1'x_2'y_1'y_2'\rangle \,.\, \mathbf{0}
\end{array}
$$

$$
\begin{array}{lll}
\xrightarrow{\tau_{\mathcal{B}}} Q_3^2 \xrightarrow{\tau_{\mathcal{B}}} & Q_3^3 = (\boldsymbol{\nu}\widetilde{m}\widetilde{n})(\boldsymbol{\nu}s_U c_U)(\boldsymbol{\nu}s_T c_T) \\
(& (\boldsymbol{\nu}w_1'z_1' :!\mathsf{bool})(\boldsymbol{\nu}w_2'z_2' :?\mathsf{bool}) \\
& (\overline{w_1'}\langle \mathsf{false}\rangle \,.\, \overline{m_6}\langle w_2'\rangle \,.\, \mathbf{0} \mid z_1'(b_1) \,.\, \overline{m_{12}}\langle \mathsf{true} z_2'b_1\rangle \,.\, \mathbf{0}) \\
& \mid n_6(s_2) \,.\, s_2(a_1) \,.\, \overline{m_7}\langle a_1\rangle \,.\, \mathbf{0} \\
& \mid n_7(a_1) \,.\, \mathbf{0} \\
& \mid n_{12}(a_1z_2b_1) \,.\, \overline{z_2}\langle a_1\rangle \,.\, \overline{m_{13}}\langle a_1b_1\rangle \,.\, \mathbf{0} \\
& \mid n_{13}(a_1b_1) \,.\, \mathbf{0} \\
& \mid \mathsf{un}\ s_U(z) \,.\, (\boldsymbol{\nu}w_1'z_1' :!\mathsf{bool})(\boldsymbol{\nu}w_2'z_2' :?\mathsf{bool})\ \overline{z}\langle w_1'w_2'z_1'z_2'\rangle \,.\, \mathbf{0} \\
& \mid \mathsf{un}\ s_T(z) \,.\, (\boldsymbol{\nu}x_1'y_1' :!\mathsf{bool})(\boldsymbol{\nu}x_2'y_2' :?(!\mathsf{bool}, ?\mathsf{bool}))\overline{z}\langle x_1'x_2'y_1'y_2'\rangle \,.\, \mathbf{0}
\end{array}
$$

$$
\begin{array}{ll}
\xrightarrow{\tau_{x_2}} & Q_4^1 = (\boldsymbol{\nu}\widetilde{m}\widetilde{n})(\boldsymbol{\nu}s_U c_U)(\boldsymbol{\nu}s_T c_T) \\
(& (\boldsymbol{\nu}w_2'z_2' :?\mathsf{bool}) \\
& (\overline{m_6}\langle w_2'\rangle \,.\, \mathbf{0} \mid \overline{m_{12}}\langle \mathsf{true} z_2'\mathsf{false}\rangle \,.\, \mathbf{0}) \\
& \mid n_6(s_2) \,.\, s_2(a_1) \,.\, \overline{m_7}\langle a_1\rangle \,.\, \mathbf{0} \\
& \mid n_7(a_1) \,.\, \mathbf{0} \\
& \mid n_{12}(a_1z_2b_1) \,.\, \overline{z_2}\langle a_1\rangle \,.\, \overline{m_{13}}\langle a_1b_1\rangle \,.\, \mathbf{0} \\
& \mid n_{13}(a_1b_1) \,.\, \mathbf{0} \\
& \mid \mathsf{un}\ s_U(z) \,.\, (\boldsymbol{\nu}w_1'z_1' :!\mathsf{bool})(\boldsymbol{\nu}w_2'z_2' :?\mathsf{bool})\ \overline{z}\langle w_1'w_2'z_1'z_2'\rangle \,.\, \mathbf{0} \\
& \mid \mathsf{un}\ s_T(z) \,.\, (\boldsymbol{\nu}x_1'y_1' :!\mathsf{bool})(\boldsymbol{\nu}x_2'y_2' :?(!\mathsf{bool}, ?\mathsf{bool}))\overline{z}\langle x_1'x_2'y_1'y_2'\rangle \,.\, \mathbf{0}
\end{array}
$$

$\xrightarrow{\tau_{\mathcal{B}}} Q_4^2 \xrightarrow{\tau_{\mathcal{B}}} \quad Q_4^3 = (\boldsymbol{\nu}\widetilde{m}\widetilde{n})(\boldsymbol{\nu}s_U c_U)(\boldsymbol{\nu}s_T c_T)$

$(\quad (\boldsymbol{\nu}w_2' z_2' :?\mathsf{bool})$

$(w_2'(a_1) \,.\, \overline{m_7}\langle a_1\rangle \,.\, \mathbf{0} \;\mid\; \overline{z_2'}\langle \mathsf{true}\rangle \,.\, \overline{m_{13}}\langle \mathsf{true\ false}\rangle \,.\, \mathbf{0})$

$\mid\; n_7(a_1) \,.\, \mathbf{0}$

$\mid\; n_{13}(a_1 b_1) \,.\, \mathbf{0}$

$\mid\; \mathsf{un}\ s_U(z) \,.\, (\boldsymbol{\nu}w_1' z_1' :!\mathsf{bool})(\boldsymbol{\nu}w_2' z_2' :?\mathsf{bool})\ \overline{z}\langle w_1' w_2' z_1' z_2'\rangle \,.\, \mathbf{0}$

$\mid\; \mathsf{un}\ s_T(z) \,.\, (\boldsymbol{\nu}x_1' y_1' :!\mathsf{bool})(\boldsymbol{\nu}x_2' y_2' :?(!\mathsf{bool}, ?\mathsf{bool}))\overline{z}\langle x_1' x_2' y_1' y_2'\rangle \,.\, \mathbf{0}$

$\xrightarrow{\tau_{x_1}} \quad Q_5^1 = (\boldsymbol{\nu}\widetilde{m}\widetilde{n})(\boldsymbol{\nu}s_U c_U)(\boldsymbol{\nu}s_T c_T)$

$(\quad \overline{m_7}\langle \mathsf{true}\rangle \,.\, \mathbf{0} \;\mid\; \overline{m_{13}}\langle \mathsf{true\ false}\rangle \,.\, \mathbf{0}$

$\mid\; n_7(a_1) \,.\, \mathbf{0}$

$\mid\; n_{13}(a_1 b_1) \,.\, \mathbf{0}$

$\mid\; \mathsf{un}\ s_U(z) \,.\, (\boldsymbol{\nu}w_1' z_1' :!\mathsf{bool})(\boldsymbol{\nu}w_2' z_2' :?\mathsf{bool})\ \overline{z}\langle w_1' w_2' z_1' z_2'\rangle \,.\, \mathbf{0}$

$\mid\; \mathsf{un}\ s_T(z) \,.\, (\boldsymbol{\nu}x_1' y_1' :!\mathsf{bool})(\boldsymbol{\nu}x_2' y_2' :?(!\mathsf{bool}, ?\mathsf{bool}))\overline{z}\langle x_1' x_2' y_1' y_2'\rangle \,.\, \mathbf{0}$

$$\xrightarrow{\tau_{\mathcal{B}}} Q_5^2 \xrightarrow{\tau_{\mathcal{B}}} Q_5^3 = (\boldsymbol{\nu}\widetilde{m}\widetilde{n})(\boldsymbol{\nu}s_U c_U)(\boldsymbol{\nu}s_T c_T)N_{P_1}$$