Parameterised Session Types

Communication Patterns Through the Looking Glass of Session Types

by

Andi Bejleri

SUBMITTED IN PART FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY IN COMPUTING OF IMPERIAL COLLEGE

Department of Computing
Imperial College of Science, Technology and Medicine

February 2012

Declaration of Originality

I hereby declare that the material presented in this dissertation is my own work, accurately referenced by background and related work.

Abstract

This dissertation studies a type theory to guarantee communication-safety in sessions of an arbitrary number of participants, typically represented as communication patterns, of mobile processes in the context of multiparty session types— a well-established type theory that describes the interactive structure of a fixed number of processes from a global point of view and type-checks the processes through projection of the global type onto the participants of the session. Communication-safety is the property that mobile processes exchange values of the same set without deadlocking and data races.

Our study introduces a programming idiom of *roles*— a concept that describes the nature of a communication pattern in a similar way to classes in Java and C#, offering a design on how to incorporate parameterised session types into a mainstream language. The formal model (1) preserves multiparty session types' syntax and type-checking strategy, and (2) allows the number of participants to range over infinite sets of natural numbers, providing full computation power of programs. A series of communication patterns and real-world examples from parallel algorithms and data exchange protocols demonstrate the expressiveness and practicality of the formal model, comparing the model with the only mature implementation of (binary) session types. We proved that type preservation under reduction and communication-safety hold in the type system.

The study of parameterised session types is supported by the examination of multiparty session types for synchronous communications. We extended the initial work on multiparty session types with a simpler calculus, multicast send of values and labels, a practical form of higher-order communication and a more intuitive, elegant linearity property; we proved that (a) type preservation and communication-safety hold in the type system, and (b) interactions of a typeable process follow exactly the description of the global type.

Acknowledgements

A PhD degree consists of the individual work of a student to identify and solve a certain problem along with the documentation of it. It is an experience that varies from one student to another and that certainly demands support, encouragement and advice from others. In my case, the following people have prepared me for such experience and helped me to go through it successfully.

The first thank you goes to my parents. To my mother, for growing in me the curiosity for science and the unknown; the warm shelter that she has provided from day one has certainly helped me to go through various situations. To my father, for giving me the hard-worker genes and the practical tips during the PhD. The proximate thank you goes to my brothers for always being there. A huge thank you goes to my grandmother, uncles and cousins for filling the days when I was home with smart humour: I have always learned from them. Lastly in this anonymity, I also want to thank my friends which I have a great deal in common. You all know who you are. A big part of what I have achieved in life is thank to you:

Urq' te Jemi perhera ne harmoni.

I want to thank my supervisor Nobuko Yoshida for the financial support and for the helpful comments on the technical material of this document. This work was partially funded by EPSRC grants EP/F003757/1 and EP/T03215/01. I am deeply grateful to Iain Phillips for helping me in circumstances where I needed supervision. He has provided effective feedback on my work and, maintained a professional and respectful relation with me. A huge thank you goes to both supervisors for reading and providing feedback several times for my dissertation. I appreciate the comments and feedback from the examiners of my viva, Susan Eisenbach and Antonio Ravara. I am thankful to many other people who have provided helpful comments on parts of this dissertation: Dave Clarke, Raymond Hu, Andrew Farrell and the anonymous reviewers. Without their contribution, this dissertation would not have been of such calibre.

I also would like to thank all the people that have encouraged, mentored and inspired me before the PhD, in chronological order: Eduard Domi, Svjetllana Kerenxhi, Ugo Montanari, Pierpaolo Degano, Jonathan Aldrich, Steffen van Bakel and Susan Eisenbach. I would like to thank Andrew Farrell, Patrick Goldsack and all the other researchers at HP Cloud Lab in Bristol for the fabulous experience spent there. A huge thank you goes to Amani El-Kholy for administrating my internship leave and return to the college. A thank you goes to all the people at Imperial which I have shared a good time at the office and at the senior common room: Raymond Hu, Rudi Ball, Dimitris Moustrous and others.

I wish to all of You the best.

Contents

List of Figures **9**

1 Introduction **11**

1.1 Guaranteeing Communication-Safety 11

1.2 Multiparty Session Types 13

1.3 Communication Patterns 18

1.4 Guaranteeing Communication-Safety for Communication Patterns 19

1.5 Parameterised Session Types 20

1.6 Thesis and Hypothesis 21

2 Synchronous Multiparty Session Types **25**

2.1 CSMS by Example 26

2.2 Synchronous Communications 28

2.3 Multicasting 29

2.4 Higher-order Communication in CSMS 31

2.4.1 Modeling in Global Types 31

2.4.2 Modeling in Processes 33

2.5 Linearity in CSMS 35

2.6 Formal Model 39

2.6.1 Syntax 40

2.6.2 Operational Semantics 44

2.6.3 Well-formedness and Linearity of Global Types 49

2.6.4 Type System 55

2.6.5 Properties of the Type System 62
2.7 Summary 82

3 Parameterised Session Types **85**
3.1 Roles 86
3.2 Communication Patterns 88
3.2.1 Tree Pattern 88
3.2.2 Star Pattern 90
3.2.3 Mesh Pattern 92
3.3 Real-World Examples 93
3.3.1 Jacobi Solution of the Discrete Poisson Equation 93
3.3.2 Dense Linear Algebra 97
3.3.3 Group Diffie-Hellman with Complete Key Authentication Protocol . 99
3.4 Formal Model 100
3.4.1 Syntax 101
3.4.2 Operational Semantics 105
3.4.3 Well-formedness of Global Types 111
3.4.4 Projection, Sorting and R-elimination 119
3.4.5 Typing Relation 126
3.4.6 Typing Communication Patterns 133
3.4.7 Properties of the Typing Relation 135
3.5 Another System 170

4 Related Work **173**
4.1 Session Types 173
4.1.1 Binary Session Types 173
4.1.2 Multiparty Session Types 176
4.1.3 Parameterised Session Types 178
4.2 Choreography and Orchestration 179
4.3 Guaranteeing Communication-Safety in Mobile Processes 180

4.4 Other Communication Systems 183

5 Conclusion **185**

5.1 Evaluation 185

5.2 Future Work 187

5.2.1 Implementing a Communication-Safe Library 188

5.2.2 Extending CSMS and CPS 189

A Synchronous Multiparty Session Types **191**

A.1 Proof of Proposition 2.6.12 191

A.2 Proof of Theorem 2.6.18 198

B Parameterised Session Types **201**

B.1 Auxiliary Definitions 201

B.2 Examples 202

B.2.1 2D-Mesh 202

B.2.2 Jacobi Solution of the Poisson Equation 203

References **205**

List of Figures

2.1 CSMS: Causality analysis. 37
2.2 CSMS: Input dependency chain. 38
2.3 CSMS: Output dependency chain. 38
2.4 CSMS: Syntax of global session types. 41
2.5 CSMS: User-defined and runtime syntax of processes. 42
2.6 CSMS: Operational semantics. 45
2.7 CSMS: Structural congruence. 46
2.8 CSMS: Well-formedness of global types. 50
2.9 CSMS: Syntax of end-point types. 55
2.10 CSMS: Typing relation for expressions and processes. 58

3.1 CPS: Diagram and global type of the Tree pattern. 89
3.2 CPS: Diagram and global type of the Star pattern. 90
3.3 Monte Carlo: Diagram of information flow. 91
3.4 CPS: Diagram and global type of the Mesh pattern. 93
3.5 CPS: Syntax of global session types. 102
3.6 CPS: Syntax of general expressions, roles and processes. 103
3.7 CPS: Operational semantics. 106
3.8 CPS: Structural congruence. 108
3.9 CPS: Well-formedness of global types. 111
3.10 CPS: Inequality between parameters. 113
3.11 CPS: Well-formedness of value and role types. 115

3.12 CPS: Syntax of role and end-point types. 119
3.13 CPS: Equality between indexes. 121
3.14 CPS: Typing system for general expressions, roles and processes. 128

B.1 CPS: Principal identifiers of a global type. 201
B.2 CPS: Free index variables of the principal that perform each action of roles. 201
B.3 CPS: Free index variables of principals. 202

Chapter 1

Introduction

1.1 Guaranteeing Communication-Safety

Almost all of the programming languages present in the software industry support communication over a dedicated environment as a primitive action to model naturally complex concurrent, distributed applications, such as web services (e.g., webmail, online sales and auctions), parallel algorithms (e.g., climate computation, algebraic computation, finding motions of particles and celestial bodies), business protocols (e.g, online banking, debit or credit card transactions, organisation between financial institutions), multi-core programming (e.g., multithreading, superscalar, vector processing), management systems for various Cloud platforms. This kind of communication, known as message-based, is modeled in programming languages by sending and receiving messages over various mediums such as sockets and memory locations. Correctly specifying message-passing communications in a mainstream language such as Java, C or C# is difficult, mainly due to the level of programming detail where there is no clear view of the conversation between the several processes. Commonly perceived disadvantages of message passing are deadlocks and unexpected message types.

For example, MPI (Gropp et al. (1999)) is a well-known, rich message-passing API to program parallel algorithms. Common MPI errors recognised by the community include:

- **Doing things before** `MPI_Init` (session-initialising construct) **and after** `MPI_Finalize` (session-closing construct). The execution of such MPI operations can lead to runtime errors such as broken invariants, unbroadcasted messages, and

incorrect collective operations.

- **Unmatched** `MPI_Send` (sending construct) **and** `MPI_Recv` (receiving construct). Such errors can lead to a mismatch between the sent and expected message type/structure, or a variety of deadlock situations depending on the communication configuration and mode. For example, two processes deadlock if each is waiting for a message before sending the message expected by the other. In the standard (buffer-blocking) mode, the converse situation (both processes attempting to send before receiving) can also deadlock: if both message sizes are bigger than the available space in the media and opposing receive buffers, then the processes cannot complete their write operations. A related problem is matching a `MPI_Bcast` output with `MPI_Recv`. Standard usage is to receive a broadcast message using the complementary `MPI_Bcast` (broadcasting construct) input. `MPI_Recv` consumes the message; hence, the receiver must be able to determine which processes have not yet seen the message and re-broadcast it manually.

- **Concurrency issues**. Incorrect access of a shared communicator (communicating media) by separate threads can violate the intended message causalities between the sender(s) and the receivers. In addition, race conditions can arise due to modifying, or even just by accessing, messages that are in transit.

These problems are crucial in message-passing parallel algorithms, web services and business protocols where the number of participants in a conversation can be of the order of tens. Recent development in networking has increased the number of connections between machines and consequently the expressivity of parallel algorithms and web services, leading to more complex sessions and making it harder for programmers to implement them correctly. The problems become more acute in the presence of conditionals and recursion.

This has motivated a vast amount of research into techniques, typically static type-systems, for guaranteeing *communication-safety* in *interactive, mobile processes* —terms that express interactive, mobile systems. We can mention behaviours by Nielson and Nielson (1994); Amtoft et al. (1997), generic types by Igarashi and Kobayashi (2004); Kobayashi et al. (2000), multiparty session types by Honda et al. (2008b), contracts by Castagna et al. (2009); Castagna and Padovani (2009) and, conversation calculus and types by Vieira et al. (2008); Caires and Vieira (2010). Type systems are a lightweight formal method that prevent programs from having non correct behaviour which is defined by some specification, and static type systems denote the class of type systems where the

process of type constraints verification happens at compile time; they are interesting to study as they detect programming errors before they become too expensive to deal with. Types in these systems capture the list of sending and receiving actions of processes in the presence of conditional and recursion (or what is called behaviour of a process) and the type constraint, guaranteeing communication-safety, is defined as in the work of Pierce (2002):

Definition 1.1.1 (Communication-safety)**.** For every "send" or "receive" on a channel of a set of values in a process, reciprocally, there is only one "receive" or "send" on the same channel of the same set of values in the other parallel processes.

1.2 Multiparty Session Types

Multiparty session types by Honda et al. (2008b) provide a type theory that addresses statically the problem of type-safe, deadlock-free interactions among a fixed number of processes. *Intuitive, light-weight, structural types* and an *efficient type-checking strategy* are the main benefits that make multiparty session types stand out from the other systems. A notion of *global type* is introduced through an intuitive syntax using simply names and arrows that describes the interaction structure between the processes —*session*— from a global point of view. The point on the intuitive syntax of global types follows from the comparison with the types of the other fore-mentioned systems, defined on bangs and question marks as we shall see later in this dissertation. Below we give the definition of session.

Definition 1.2.1 (Session)**.** A session is the interactive structure formed by the "sending-receiving" actions in the presence of conditionals and recursion, between communicating processes. It has an identity ensuring that the sequence of interactions belonging to a session is not confused with other concurrent runs.

Multiparty session types have a light-weight type annotation of programs since only one global type types a program, independently of the number of processes; i.e., there is one global type for a program of two processes, one global type for a program of ten processes. The other systems, except generic types, build on a heavy-weight type annotation —a type for each process— increasing the effort of programming; i.e., in addition of defining a program of two processes, a programmer defines also two process types and for a program

of ten processes, ten process types. Generic types do not suffer from this shortfall as Kobayashi (2006) has introduced a general type inference algorithm, building the system on a variant of the π-calculus without type annotation. However, by having no types, generic types do not alleviate the effort of programming and understanding communication-based programs. In the next section, we illustrate the relevance of having a level of abstraction on top of mainstream languages that (1) describes every component of the structure of interactions and (2) makes programmers step away from the mainstream language and understand the goal of a program.

By specifying the *structure of interactions between the participants of a session*, global types provide a smart mode of organising communications that allows the description of their *order*. Order of communication is part of the program logic and so, is an integral part of the program development. An incorrect implementation of it affects the semantics of a program and may therefore lead to broken invariant as we shall see in the next section.

We shall illustrate the fore-mentioned benefits through an example of a possible Automatic Teller Machine (ATM), Debit Card Agency and Bank specification. A formal validation will be given throughout this dissertation.

An Example: ATM–Debit-Card–Bank

A debit card owner inserts their card into the ATM terminal and enters the PIN code. The ATM communicates with the issuer of the debit card to verify the pin number. If correct, the client can carry out one of the three standard operations: 1) withdraw money, 2) check the account balance or 3) make a deposit. If incorrect, the client is unable to proceed. The global description of the conversation in a name-arrow based representation —global type, including end-points, type of message and order of communcations— is:

$$
\begin{aligned}
G = \;& \texttt{ATM} \to \texttt{Debit-Card Issuer}\colon 1\ \langle int \rangle. \\
& \texttt{Debit-Card Issuer} \to \texttt{ATM}\colon 1\ \{ \\
& \qquad ok\colon \texttt{ATM} \to \texttt{Bank}\colon 2 \\
& \qquad\qquad \{withdraw\colon \texttt{ATM} \to \texttt{Bank}\colon 2\ \langle int \rangle.\texttt{end}, \\
& \qquad\qquad\ balance\colon \texttt{Bank} \to \texttt{ATM}\colon 2\ \langle int \rangle.\texttt{end}, \\
& \qquad\qquad\ deposit\colon \texttt{ATM} \to \texttt{Bank}\colon 2\ \langle int \rangle.\texttt{end}\}, \\
& \qquad ko\colon \texttt{end}\}
\end{aligned}
$$

where $\texttt{A} \rightarrow \texttt{B}\colon \mathrm{m}\langle U \rangle$ means that participant `A` sends a message of type U to participant `B` through channel m, $\texttt{A} \rightarrow \texttt{B}\colon \mathrm{m}\{l_1\colon \cdots, l_2\colon \cdots, l_3\colon \cdots\}$ means that participant `A` sends a label l_i where $i \in \{1, 2, 3\}$ to participant `B` through channel m and the conversation follows that path, and `end` signifies the end of a conversation. We refer to $\texttt{A} \rightarrow \texttt{B}\colon \mathrm{m}$ throughout this dissertation as *causality*. The first causality, $\texttt{ATM} \rightarrow \texttt{Debit}-\texttt{Card Issuer}$, describes the delivery of the pin number (integer) from the ATM to the issuer of the Debit Card. In the second causality, $\texttt{Debit}-\texttt{Card Issuer} \rightarrow \texttt{ATM}$, the issuer of the Debit Card internally checks the pin number and subsequently informs the ATM about the outcome through a label *ok* or *ko*, representing respectively the case when the pin is correct or incorrect. For the *ok* label, the ATM chooses one of the three labels *withdraw*, *balance* and *deposit* and informs the Bank as described in the causality $\texttt{ATM} \rightarrow \texttt{Bank}$; otherwise, for the *ko* label, the conversation ends. The three paths of the conversation are: (1) for the *withdraw* label: the ATM sends the amount of money (integer) to the Bank as denoted in the causality $\texttt{ATM} \rightarrow \texttt{Bank}$, (2) for the *balance* label: the Bank sends to the ATM the amount of the client's savings as denoted in the causality $\texttt{Bank} \rightarrow \texttt{ATM}$, and (3) for the *deposit* label: the ATM sends the amount of money deposited by the client to the Bank as denoted in the causality $\texttt{ATM} \rightarrow \texttt{Bank}$. This higher, structured view of programming makes the programmer step away from the mainstream language and focus only on the communication components: participants of a session, message types and order of communications. Also, global types help programmers understand the goal of a program, avoiding the necessity to test the code in the event of a mis-sent communication.

We shall explain the point on the intuitive and structural syntax of global types by defining this example in the syntax of conversation types— another system that describes conversations from a global point of view, in contrast to the others which describe only interactions of single processes (similarly to our end-point types as we shall see later). That is:

$$\begin{aligned} G = \quad & \tau pin(int).\oplus\{\tau ok().\oplus\{ \\ & \qquad \tau withdraw().\tau money(int).\mathbf{0}, \\ & \qquad \tau balance().\tau money(int).\mathbf{0}, \\ & \qquad \tau deposit().\tau money(int).\mathbf{0}\}, \\ & \tau ko().\mathbf{0}\} \end{aligned}$$

where $\tau m(U)$ means an internal action (τ) of message m and type U, and $\oplus\{\tau \mathrm{m}_1()..., \tau \mathrm{m}_2()..., ...\tau \mathrm{m}_n()...\}$ means a branching of messages $\mathrm{m}_1, \mathrm{m}_2, ..., \mathrm{m}_n$. The above definition lacks communication components such as participants, making it difficult to understand the

conversation structure. In addition, the conversation type lacks constructs that are at the core of designing structured interactions such as branching over labels as we shall see later in the next chapter. These two shortfalls do not help programmers to define interaction structures that hold invariants and understand the goal of a program. Indeed, consider the following implementation of the ATM, where the logic specifying that the ATM communicates with the issuer of the debit card to verify the pin number is defined in the Bank code:

$$
\begin{aligned}
&ATM \triangleq \overline{a}_{[2]}\,(x_1).\overline{b}_{[2]}\,(x_2).\\
&\quad x_1!\langle pin\rangle;\texttt{if } withdraw \texttt{ then } x_2 \oplus \langle withdraw\rangle; x_2!\langle amount\rangle;\\
&\qquad\qquad \texttt{else if } balance \texttt{ then } x_2 \oplus \langle balance\rangle; x_2?(y);\\
&\qquad\qquad\qquad \texttt{else if } deposit \texttt{ then } x_2 \oplus \langle deposit\rangle; x_2!\langle amount\rangle;
\end{aligned}
$$

where a and b denote public access points that serve to initiate sessions, $x!\langle v\rangle$ denotes the action of sending a message v through channel x and $x?(y)$ denotes the action of receiving a message place-hold by y through channel x. This implementation is well-typed, since global conversation types do not specify the participants of a communication, allowing the programmer to incorrectly define this part of the program logic within the Bank code instead of the ATM code. Caires and Vieira (2010) alleviate the lack of participants in global conversation types by adding process types to the program definition. However, programmers can write process types that well-type the fore-mentioned behavior, since the coherence between global and process types cannot be defined on participants. The other systems mentioned in Section 1.1 and binary session types of Honda et al. (1998) do not provide a structural definition of the interactions from a global point of view and so, do not express their order. In the ATM example, those systems well-type the ATM that operates transactions with the Bank before the Debit Card company has verified its identity through the pin number defined as:

$$
\begin{aligned}
&ATM \triangleq \overline{a}_{[2]}\,(x_1).\overline{b}_{[2]}\,(x_2).\\
&\quad x_1!\langle pin\rangle;\texttt{if } withdraw \texttt{ then } x_2 \oplus \langle withdraw\rangle; x_2!\langle amount\rangle;\}x_1 \rhd \{ok\colon \mathbf{0}, ko\colon \mathbf{0}\}\\
&\qquad\qquad \texttt{else if } balance \texttt{ then } x_2 \oplus \langle balance\rangle; x_2?(y); x_1 \rhd \{ok\colon \mathbf{0}, ko\colon \mathbf{0}\}\\
&\qquad\qquad\qquad \texttt{else if } deposit \texttt{ then } x_2 \oplus \langle deposit\rangle; x_2!\langle amount\rangle; x_1 \rhd \{ok\colon \mathbf{0},\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad ko\colon \mathbf{0}\}\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad \texttt{else } x_1 \rhd \{ok\colon \mathbf{0}, ko\colon \mathbf{0}\}
\end{aligned}
$$

where $x \rhd \{l_1 : ..., l_2 : ...., l_n : ...\}$ denotes the action of receiving one of the label $l_i, i \in \{1, ..., n\}$ through channel x; this construct comes from the calculus of session types

(both binary and multiparty), which can easily be encoded in the calculus of the other systems. The types used to type-check the above process, including the types for the other processes, in the syntax of binary session types which similar to the others, are:

$$\text{ATM-Bank} \triangleq 1!\langle int\rangle; 1 \rhd \{ok\colon \mathsf{end}, ko\colon \mathsf{end}\}$$
$$\text{ATM-Debit Card} \triangleq x_2 \oplus \{withdraw : 2!\langle int\rangle; \mathsf{end}, balance : 2?(int); \mathsf{end}, deposit : 2!\langle int\rangle; \mathsf{end}\}$$
$$\text{Bank-ATM} \triangleq 1?(int); 1 \oplus \{ok\colon \mathsf{end}, ko\colon \mathsf{end}\}$$
$$\text{Debit Card-ATM} \triangleq x_2 \rhd \{withdraw : 2?(int); \mathsf{end}, balance : 2!\langle int\rangle; \mathsf{end}, deposit : 2?(int); \mathsf{end}\}$$

Communication-safety holds for both sessions Bank-ATM, ATM-Bank and ATM-Debit Card, Debit Card-ATM: there is a "receive" on the complementary process for each "send" on one process and vice versa. However, the type system does not capture the incorrect order of communications between ATM-Bank and ATM-Debit Card. This implementation is ill-typed by the multiparty session types as global types capture the structure of communications between all participants of the session. Also, the syntax of the above types is similar to the one of processes, using the same constructs such as: bang, question mark, o plus and right-hand-drive, and so not allowing programmers to step away from the programming language.

Despite the strong benefits, multiparty session types are not composable, restricting the expressiveness of the language when it comes to interleaving of sessions. That is, legal programs that consist of processes participating in two or more sessions may be ill-typed. For example, consider the following program of two processes initiating two sessions, exchanging two integers over the first one and a boolean value over the second (Castagna and Padovani (2009)):

$$\text{P} = \overline{a}_{[2]}(x).\overline{b}_{[2]}(y).x!\langle 3\rangle;;y!\langle true\rangle; x?(z); 0$$
$$\text{Q} = a_{[2]}(x).b_{[2]}(y).x?(z); y?(w); x!\langle 5\rangle; 0$$

This program is ill-typed by the type-system of Bettini et al. (2008), since no partial order between sessions a and b, which define also the usage of session channels, can be defined (The system of Bettini et al. (2008) is another variant of multiparty session types by Honda et al. (2008b) where channels are omitted from the syntax of global types, serving a simpler and more expressive type system than the original one). In a system of contracts (Castagna and Padovani (2009)), this program is well-typed as a result of the typing strategy on processes rather than on channels as session types. This result is valid

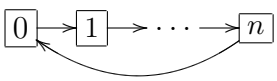
also for the other systems mentioned at the end of Section 1.1.

Multiparty session types are based on a variant of the π-calculus of Milner et al. (1992) with asynchronous communications extended with primitives for structuring communications: branching of a process behaviour into several paths, conditional through the "if-else" construct and repetitive behaviour through recursive functions. Type-checking of processes is obtained through *projection* of global types onto participants of a session; projection is a key element in the formal model of the system and will be explained in the next chapter. Thus, global types serve not only as a human-interpretable descriptive language of a system's architecture but also as a type system for processes.

1.3 Communication Patterns

Communication patterns describe simple and elegant structured interactions in communication-based applications. They are used in many parallel computing architectures of parallel algorithms (Gropp et al. (1999); Leighton (1991)), data exchange protocols (Ateniese et al. (1998)) and web services (Web Services Choreography Working Group (2005)). Communication patterns, as design patterns (Gamma et al. (1995)), help programmers to design more modular and more user-friendly system architectures. In parallel algorithms, communication patterns define the assignment of processes to regions of the problem domain. Hence, the choice of the pattern affects the performance of the algorithm. Common communication patterns are Ring, Star, Tree, Mesh and Hypercube.

For example, the Ring pattern is used to implement the Simulation of the n-Body algorithm— finding the motion, in accordance with classical mechanics, of a system of particles given their masses, initial positions and velocities. Each participant communicates with exactly two neighbours: participant j communicates with participants $j-1$ and $j+1$ $(1 \leq j \leq n-1)$, with the exception of 0 who communicates with n and 1, and n with $n-1$ and 0 as illustrated in the diagram below.



Each simulation involves $n-1$ iterations. In the first iteration, each process sends their particle data to its neighbour on the left and calculates the partial resultant forces exerted

within their own particle set. In the i-th iteration, each process forwards on the particle data received in the previous iteration, adds this data to the running force calculation, and receives the next data set. The particle data from the left neighbour is received by the end of the final iteration and so, each data set has now been seen by all processes in the Ring, allowing the final position of the particles for the current simulation to be calculated.

Other communication patterns such as Star, 1D- and 2D-Mesh are used in the design of other data exchange protocols, parallel algorithms, e.g., Dense Linear Algebra, Group Diffie-Hellman with Complete Key Authentication Protocol and Jacobi solution of the Poisson Equation as we shall see later in this dissertation.

1.4 Guaranteeing Communication-Safety for Communication Patterns

Both communication patterns and global types describe structured interactions, and the latter provides not only a blue-print of the system architecture but also a type system that guarantees type-safe, deadlock-free interactions in the system's implementation. At this point, a question arises as to how to specify all instances of a communication pattern in a single global type, so that programmers can benefit from the type theory.

For example, the Ring pattern of two participants is specified in the global type syntax as $\mathtt{0} \to \mathtt{1}{:}\langle U\rangle.\mathtt{1} \to \mathtt{0}{:}\langle U\rangle.\mathsf{end}$ and for three participants as $\mathtt{0} \to \mathtt{1}{:}\langle U\rangle.\mathtt{1} \to \mathtt{2}{:}\langle U\rangle.\mathtt{2} \to \mathtt{0}{:}\langle U\rangle.\mathsf{end}$. Building global types of other instances given the number of participants is relatively easy. Unfortunately, in parallel algorithms and other communication-based applications the number of participants is known only at run-time; e.g. in parallel algorithms, the number of processes assigned to compute the answer of a problem instance is in proportion to its size. In the n-Body algorithm described in the previous section, 80 processes are required to achieve a good speed of the algorithm for 64 particles and 120 processes are required for 128 particles for the same speed up as demonstrated by Pereira (1998)[1]. We would like a type theory of global types that answers this research question:

> *How can programmers specify a single global type that captures all the instances of a communication pattern which have a different number of participants?*

[1]Pereira uses a different algorithm from the one presented in this dissertation; thus, the ratio of processes and number of particles belongs to that algorithm.

Another problem is the design of processes that implement the behaviour of each participant. For example, the behaviour of `1` in the Ring pattern of two participants is: receive from `0` and then send to `0`, and of three participants: receive from `0` and then send to `2`. As in global types, building processes of other instances given the number of participants of a pattern is relatively easy. Our formal model needs to address also a second research question:

> *How can programmers specify a single program that captures all the instances of a communication pattern which have a different number of participants?*

Our solution to the two research questions extends the multiparty session type theory with parameters and introduces a programming idiom of *roles*, briefly stated in the next section.

1.5 Parameterised Session Types

The solution proposed to the first problem is to parameterise participants, to iterate over parameterised causalities that abstract the repetitive behaviour of a pattern, and to compose sequentially global types. We use the **R** operator from Gödel's theory T (Alves et al. (2010)) of primitive recursive functions to formalise the three idioms. For example, in the Ring pattern, the causality that abstracts the communications from 0 to n in the diagram is $\mathtt{i} \rightarrow \mathtt{i{+}1}{:}\langle U \rangle$, where `i` and $\mathtt{i}+1$ are parameterised participants or *principals*, as we will refer to them throughout this dissertation, and $0 \leq \mathtt{i} \leq n-1$ and $n \geq 1$. Given the number of participants (n), the **R** operator will iterate over the parameterised causality, and then the global type created will be composed with the causality $\mathtt{n} \rightarrow \mathtt{0} : \langle U \rangle$ to complete an instance of the Ring. The **R** operator can be understood through the two reduction rules:

$$\begin{aligned} \mathbf{R}\ G\ \lambda i.\lambda \mathbf{x}.G'\ 0 &\longrightarrow G \\ \mathbf{R}\ G\ \lambda i.\lambda \mathbf{x}.G'\ (\mathrm{n}+1) &\longrightarrow G'\{\mathrm{n}/i\}\{(\mathbf{R}\ G\ \lambda i.\lambda \mathbf{x}.G'\ \mathrm{n})/\mathbf{x}\} \end{aligned}$$

For each natural number, we obtain a global type by applying the two rules. In each iteration, the index variable in G' is substituted by a predecessor of n+1 and $\mathbf{x}$ is replaced by instances of the parameterised causalities present in G', except 0 when $\mathbf{x}$ is replaced by instances of G. The type system allows values of parameters that abstract the number of

participants to range over infinite sets of natural numbers, providing full computation power of programs that implement parameterised communication patterns. The point on the full computation power follows from the comparison of another work that models parameterised session types by Yoshida et al. (2010), where values of parameters range over finite sets of natural numbers, restricting the number of participants in a communication pattern and consequently the computation power of the program implementing the particular pattern. A comparison of our system and the fore-mentioned is discussed later in this dissertation.

The solution proposed to the second problem introduces the syntax of roles that includes the **R** operator to parameterise participants, to iterate over roles that abstract the same behaviour that runtime processes share and to compose in parallel roles. In the Ring, there are three kinds of participants: the first one is `0` which sends to the participant on his left (`1`) and then receives from the last participant (n), the second one is `i` for $1 \leq \texttt{i} \leq n-1$ that receives from the participants on his right (`i−1`) and then sends to the one on his left (`i+1`) and finally, the last participant n, which receives from the participant on his right ($n-1$) and then sends to the first participant (`0`). There is a role for each of the three kinds of participant. The **R** operator will iterate over the role of `i`, returning on each iteration processes that share the same behaviour, and then will compose them in parallel with the processes of the roles of `0` and n.

1.6 Thesis and Hypothesis

The thesis of this dissertation is stated:

> *Parameterised session type theory can be used to guarantee communication-safety in sessions of an arbitrary number of participants, typically represented as communication patterns, of mobile processes, supporting (1) a concept similar to class—role, (2) an intuitive, structural, light-weight type annotation of programs and (3) an efficient, liberal typing.*

This thesis is based on the following hypotheses, presented along with the evidence that support them.

Hypothesis I A simple, practical calculus of multiparty sessions, based on the π-calculus, can be used to model structured interactions between a fixed number of participants,

employing an intuitive, structural, light-weight type annotation.

Validation

1. The Calculus for Synchronous Multiparty Sessions (CSMS) design and syntax preserves the type annotation of the initial work on multiparty session types: Sections 2.1, 2.2 and 2.6.1.

2. Multicast send of values and labels increases expressivity of multiparty sessions and minimises the effort of programming: Section 2.3.

3. A practical form of higher-order communication models safely the capability of a process to delegate its participation in a session to another one without letting the other participants know: Section 2.4.

4. An intuitive, elegant linearity property ensures that two different communications using the same channel do not have data races (The point on the elegant, intuitive definition of linearity follows from the comparison with the definition of Honda et al. (2008b)): Sections 2.5 and 2.6.3.

Hypothesis II An efficient type-checking strategy can be achieved through the projection of global types onto participants of a session to guarantee communication-safety.

Validation

1. CSMS type system and complexity analysis of global types coherence algorithm in cubic computational steps: Section 2.6.4 and Appendix A.2.

2. Proof of type preservation, communication-safety and session fidelity of CSMS type system: Section 2.6.5.

Hypothesis III By extending multiparty session types to parameterised session types, richer patterns of communication and programming constructs can be expressed naturally and safely, and control of the main source of programming errors in MPI (Gropp et al. (1999)) is achieved.

Validation

1. Examples that show how to present various communication patterns and express the *for* loop, and control the index calculation problem in MPI: Section 3.2.

2. Real-world examples from parallel algorithms and a key distribution protocol: Section 3.3.

Hypothesis IV The idiom of roles offers a design on how to incorporate the theory of parameterised session types into class-based languages such as Java and C#.

Validation

1. Role design and syntax of the Calculus of Parameterised Sessions (CPS): Section 3.1 and 3.4.1.

Hypothesis V A global type annotation and an efficient, liberal type-checking strategy can be achieved by extending the projection algorithm of multiparty session types to parameterised participants (principals) to guarantee communication-safety.

Validation

1. CPS type system preserves the type-checking strategy of multiparty session types through three elegant, practical concepts: (1) projection, (2) sorting and (3) **R**-elimination, performing the process in loglinear time, and allows values of parameters to range over infinite sets of natural numbers: Sections 3.4.4 and 3.4.5.

2. The point on the liberal type system follows from the comparison with the type system of Yoshida et al. (2010): Section 3.5.

3. Proof of type preservation and communication-safety of CPS type system: Section 3.4.7.

Publications

Part of this work was published in the following papers with the description of the author's contribution:

- Andi Bejleri and Nobuko Yoshida. Synchronous Multiparty Session Types. In International Workshop on Programming Language Approaches to Concurrency and Communication-cEntric Software (PLACES 08), Electr. Notes Theor. Comput. Sci., 241:3-33 (2009).

  Author's contribution: The material of this paper is an account of my investigation into Nobuko Yoshida's proposal to study multiparty session types for synchronous communications and multicasting.

- Andi Bejleri, Raymond Hu and Nobuko Yoshida. Session-based Programming for Parallel Algorithms. In International Workshop on Programming Language Approaches to Concurrency and Communication-cEntric Software (PLACES 09), EPTCS, 17:17-29 (2009).

  Author's contribution: The material of this paper is an account of my investigation into my proposal to evaluate session programming and typing through parallel algorithms. Raymond Hu's contribution consists of the design and implementation of SJ, notion of transport-independence, writing the introduction and editing the text related to SJ features in the other sections with Nobuko Yoshida.

- Andi Bejleri. Practical Parameterised Session Types. In Proceedings of International Conference on Formal Engineering Methods (ICFEM 10), Lect. Notes Comput. Sci., 6447:270-286 (2010).

  Author's contribution: The material of this paper is an account of my investigation into my proposal to study a global type theory for sessions of an arbitrary number of participants.

Chapter 2

Synchronous Multiparty Session Types

This chapter introduces the Calculus of Synchronous Multiparty Sessions: CSMS and its type system. CSMS is a variant of the π-calculus extended with primitives for structuring interactions. CSMS is based on the initial calculus of multiparty sessions by Honda et al. (2008b) and extends the latter by introducing: multicast send of messages and a practical form of higher-order communication. The type system of CSMS extends the session type theory of the initial system for synchronous communications. Differences between this system and the initial one will be discussed throughout this chapter.

Section 2.1 gives the intuition of programming in CSMS through an example: a distributed version of the addition operation on two natural numbers. Section 2.2 discusses how synchronous communications naturally capture behaviours where control on timing and strong sequential order of communication are essential to the program specification. Section 2.3 gives the intuition of multicast send of messages and discusses its benefits. Higher-order communication is described in Section 2.4, discussing differences between this system and the initial one. Linearity of channels in CSMS is introduced in Section 2.5, describing differences between the initial version of multiparty sessions. The formal model of CSMS, including syntax of global types and processes, operational semantics, well-formedness and linearity of channels in global types, type system and properties of the latter is given in Section 2.6 and finally Section 2.7 concludes.

2.1 CSMS by Example

A user-defined program in CSMS is a global type and a fixed number of processes, while a run-time program is a fixed number of processes composed in parallel forming a session. The global type describes the interaction structure between the processes of a session, defined by the "sending-receiving" actions in the presence of conditionals and recursion, from a global point of view. Each process of the session defines a behaviour that is different from the others. A process behaviour typically starts with a session initiation prefix over a session identifier, followed by a sequence of sending and receiving actions. The sequence of communications happens privately inside the session; i.e. the messages sent by a process are read only by processes of the same session. The behaviour of a process can branch over several paths prefixed by labels where the decision to choose a path is defined internally by the process logic or externally by other processes through communication primitives. Also, a process can iterate infinitely over a behaviour.

Programming Methodology The programming methodology of multiparty sessions follows a top-down approach. The **first step** when programming a multiparty session is the definition of the global type for the intended conversation. In the **second step**, the programmer programs each process of the session. Programmers that define the global type and processes may be different.

The following example of a distributed version of the addition operation on two natural numbers gives an informal introduction to CSMS. The *Addition* session is defined over four different participants, namely `Client`, `Addition`, `Successor` and `Predecessor`. In that session, `Client` sends two natural numbers to `Addition` and waits to receive the sum. `Addition` checks whether the second operand is equal to 0. For a second operand of value 0, `Addition` sends the notification of answer and first operand as result to `Client`, otherwise it sends the first and second operands respectively to `Successor` and `Predecessor`, and receives from them respectively the successor of the first and predecessor of the second operands; this behaviour is repeated until the second operand equals 0. The global description of the conversation in the global type syntax is:

$$
\begin{aligned}
&\texttt{Client} \to \texttt{Addition}\colon\ 1\langle int, int\rangle.\\
&\mu \texttt{t}.\texttt{Addition} \to \{\texttt{Successor}, \texttt{Predecessor}\}\colon 2,3\{\\
&\qquad\qquad true : \texttt{Addition} \to \texttt{Client}\colon\ 1\{\\
&\qquad\qquad\qquad\qquad answer : \texttt{Addition} \to \texttt{Client}\colon\ 1\langle int\rangle.\texttt{end},
\end{aligned}
$$

$$\begin{aligned} false : \texttt{Addition} \rightarrow \texttt{Successor}\colon\ & 2\langle int\rangle. \\ & \texttt{Addition} \rightarrow \texttt{Predecessor}\colon\ 3\langle int\rangle. \\ & \texttt{Successor} \rightarrow \texttt{Addition}\colon\ 2\langle int\rangle. \\ & \texttt{Predecessor} \rightarrow \texttt{Addition}\colon\ 3\langle int\rangle.\mathbf{t}\} \end{aligned}$$

where $\texttt{A} \rightarrow \{\texttt{B},\texttt{C}\}\colon \mathrm{m}_1, \mathrm{m}_2\{l_1\colon \cdots, l_2\colon \cdots\}$ means that participant A sends simultaneously a label l_i where $i \in \{1,2\}$ to participants B and C through channels m_1 and m_2; the label *true* represents the case when the second operand equals 0, while the label *false* denotes otherwise. Infinite behaviour is defined through the μ operator.

The processes of the session, namely *client*, *addition*, *successor* and *predecessor*, are given below. The *client* process sends a multicast session request to the other processes through the shared name a— a public point of access for all the participants which identifies the session. Participants in the definition of processes are defined by natural numbers: `Client` as `1`, `Addition` as `2`, `Successor` as `3` and `Predecessor` as `4`. When all the four processes are present, then fresh session channels are generated at runtime and substituted into placeholders (x_1, x_2, x_3) introduced in the processes scope. *client* starts the interaction by sending the multiple value (5, 4) on channel x_1 ($x_1!\langle 5,4\rangle;$). Subsequently, *addition* receives the value on the same channel ($x_1?(y_1, y_2);$) and invokes an iterative behaviour ($\mathsf{def}\, X_1\langle y_1, y_2, x_1, x_2, x_3\rangle = ...$) that checks whether the value of the second argument is equal to 0 or not. For each branch chosen, a label is sent in multicast through channels x_2, x_3 ($x_2, x_3 \oplus \langle ...\rangle$); multicast will be discussed in more detail later in Section 2.3. The label is received by *successor* and *predecessor* on respectively channels x_2 and x_3 ($x_2\&\{...\}$ and $x_3\&\{...\}$) and the conversation follows the chosen path. The reduction steps of the program are given in Section 2.6.2.

$$client \triangleq \overline{a}_{[2,3,4]}\,(x_1, x_2, x_3).x_1!\langle 5,4\rangle; x_1\&\{answer : x_1?(y_1); 0\}$$

$$\begin{aligned} addition \triangleq\ & \mathsf{def}\ X_1\langle y_1, y_2, x_1, x_2, x_3\rangle = \\ & \qquad \texttt{if}\ (y_2 = 0)\ \texttt{then}\ x_2, x_3 \oplus \langle true\rangle; x_1 \oplus \langle answer\rangle; x_1!\langle y_1\rangle; P \\ & \qquad \texttt{else}\ x_2, x_3 \oplus \langle\ false\rangle; x_2!\langle y_1\rangle; x_3!\langle y_2\rangle; x_2?(y_1); x_3?(y_2); \\ & \qquad\qquad X_1\langle y_1, y_2, x_1, x_2, x_3\rangle \\ & \mathsf{in}\ a_{[2]}\,(x_1, x_2, x_3).x_1?(y_1, y_2); X_1\langle y_1, y_2, x_1, x_2, x_3\rangle \end{aligned}$$

$$\begin{aligned} successor \triangleq\ & \mathsf{def}\ X_2\langle x_2\rangle = x_2\&\{true\colon \mathbf{0}, false\colon x_2?(y); x_2!\langle y+1\rangle; X_2\langle x_2\rangle\} \\ & \mathsf{in}\ a_{[3]}\,(x_1, x_2, x_3).X_2\langle x_2\rangle \end{aligned}$$

$$\begin{aligned} predecessor \triangleq\ & \mathsf{def}\ X_3\langle x_3\rangle = x_3\&\{true\colon \mathbf{0}, false\colon x_3?(y); x_3!\langle y\rangle; X_3\langle x_3\rangle\} \\ & \mathsf{in}\ a_{[4]}\,(x_1, x_2, x_3).X_3\langle x_3\rangle \end{aligned}$$

2.2 Synchronous Communications

Synchronous communications are useful to model multiparty sessions where control for timing events and strong sequential order of messages are essential to the problem specification. Indeed, the runtime sequence of interactions for synchronous communications follows more strictly the one of the global type than for asynchronous communications; e.g. in the global type $\mathtt{A} \rightarrow \mathtt{B} : \mathrm{m}.\mathtt{A} \rightarrow \mathtt{C} : \mathrm{n}$, the runtime sequence of synchronous communications will follow the order of communications as in the global type: `A` sends the second message to `C` after `B` has received the first message while the runtime sequence of asynchronous communications may have `C` receiving the message before `B`, breaking the order of causalities in the global type. We shall illustrate the benefit through the modeling of a real-world example.

Consider a fire alarm system that alerts and safely evacuates personnel from a building in case of fire. We expect that in case of fire all fire alarms sound before the elevators block (NFPA 72, fire alarm code). This scenario would be modeled by a control process that sends in multicast an *ON* message to all fire alarms and a *BLOCK* message to all elevators. The global type that describes this behaviour is defined below:

$$\mathtt{Controller} \rightarrow \mathtt{FireAlarm}_{\mathtt{1}}, ..., \mathtt{FireAlarm}_{\mathtt{j-1}} : 1, ..., j-1\langle string \rangle.$$
$$\mathtt{Controller} \rightarrow \mathtt{Elevator}_{\mathtt{j}}, ..., \mathtt{Elevator}_{\mathtt{j+k-1}} : 2*j, ..., 2*j+k-1\langle string \rangle$$

The timing of events in this example can be obtained by modeling the session using synchronous communications. In CSMS, the second "send" will take place only after the first message has been received by all the fire alarms; the implementation below follows correctly the timing specification of the events in a fire alarm system.

$$controller \triangleq \overline{a}_{\mathtt{[2,3,...,j+k]}}(x_1, x_2, ..., x_{j-1}, y_j, ..., y_{j+k-1}).x_1, ..., x_{j-1}!\langle \text{“}ON\text{”} \rangle;$$
$$y_j, ..., y_{j+k-1}!\langle \text{“}BLOCK\text{”} \rangle; P$$
$$firealarm_1 \triangleq a_{\mathtt{[2]}}(x_1, x_2, ..., x_{j-1}, y_j, ..., y_{j+k-1}).x_1?(x); P_1$$
$$\ldots$$
$$firealarm_{j-1} \triangleq a_{\mathtt{[j]}}(x_1, x_2, ..., x_{j-1}, y_j, ..., y_{j+k-1}).x_{j-1}?(x); P_{j-1}$$
$$elevator_j \triangleq a_{\mathtt{[j+1]}}(x_1, x_2, ..., x_{j-1}, y_j, ..., y_{j+k-1}).y_j?(x); Q_1$$
$$\ldots$$
$$elevator_{j+k-1} \triangleq a_{\mathtt{[j+k]}}(x_1, x_2, ..., x_{j-1}, y_j, ..., y_{j+k-1}).y_{j+k-1}?(x); Q_{k-1}$$

For asynchronous communications as in Honda et al. (2008b), the timing of events is modeled by hardcoding the global description, adding extra communications that inform the controller that fire alarms have received the message. Thus, the global type that

assures timing of events in Honda et al. (2008b) is defined as:

$$\begin{array}{l}
\texttt{Controller} \rightarrow \texttt{FireAlarm}_1, ..., \texttt{FireAlarm}_{\texttt{j}-1} : 1, ..., j-1\langle string\rangle. \\
\texttt{FireAlarm}_1 \rightarrow \texttt{Controller} : j\langle\rangle. \\
\ldots \\
\texttt{FireAlarm}_{\texttt{j}-1} \rightarrow \texttt{Controller} : 2*j-1\langle\rangle. \\
\texttt{Controller} \rightarrow \texttt{Elevator}_{\texttt{j}}, ..., \texttt{Elevator}_{\texttt{j+k}-1} : 2*j, ..., 2*j+k-1\langle string\rangle
\end{array}$$

where causalities $\texttt{FireAlarm}_1 \rightarrow \texttt{Controller}$, ..., $\texttt{FireAlarm}_{\texttt{j}-1} \rightarrow \texttt{Controller}$ are added after the first causality to notify the controller that the fire alarms have received the message and so, the controller can subsequently send the second message to the elevators, following the timing of events as described in the specification.

2.3 Multicasting

CSMS supports the delivery of a message to a group of participants simultaneously. Multicast send of values is a natural communication idiom present in many scenarios in technology and everyday life: it is very common that we find ourselves speaking to a group of persons at a dinner party. In the distributed version of addition, the stopping condition —the second operand is equal to 0— must be communicated to both `Successor` and `Predecessor` from `Addition` as:

$$\texttt{Addition} \rightarrow \{\texttt{Successor}, \texttt{Predecessor}\}2, 3\{true : ....false : ...\}$$

The fore-mentioned behaviour can be modeled without multicasting as:

$$\begin{array}{r}
\texttt{Addition} \rightarrow \texttt{Successor}\colon 2\{true : \texttt{Addition} \rightarrow \texttt{Predecessor}\colon 3\{true : ...\}, \\
false : \texttt{Addition} \rightarrow \texttt{Predecessor}\colon 3\{false :\}\},
\end{array}$$

resulting in a verbose global type definition.

Multicast send of labels increases expressivity of multiparty session types, describing behaviours previously prohibited by projection of global types on branching as we shall see in Section 2.6.4. For example, the global type:

$$\texttt{A} \rightarrow \texttt{B} : \mathrm{m}_1\{l_1 : \texttt{B} \rightarrow \texttt{A} : \mathrm{m}_2\langle U\rangle,\ l_2 : \texttt{C} \rightarrow \texttt{A} : \mathrm{m}_2\langle U'\rangle\}$$

describes an undesirable behaviour (non-coherent by projection) as C sends a message of type U' on channel m_2 independently to the label participant A chooses, leading to race conditions and type errors in the case when process A chooses the first label l_1. We can safely model this behaviour by defining A sending in multicast to both B and C the label chosen so that depending on the label chosen by A, one of the two processes B and C sends a message to A on channel m_2. The global type that describes this behaviour is defined as:

$$\mathtt{A} \to \{\mathtt{B}, \mathtt{C}\} : \{\mathrm{m}_1, \mathrm{m}_3\}\{l_1 : \mathtt{B} \to \mathtt{A} : \mathrm{m}_2\langle U\rangle,\ l_2 : \mathtt{C} \to \mathtt{A} : \mathrm{m}_2\langle U'\rangle\}.$$

We shall further evaluate the expressivity of multicasting in the two buyers and seller protocol given by Honda et al. (2008b). Two buyers need to buy a book together and each of them pays a part of the book price. The seller informs simultaneously the two buyers about the price of the book. When receiving the price, the first buyer informs the second buyer of how much he owes. Upon that value, the second buyer checks whether he can afford to pay his share. In both cases, the second buyer simultaneously informs both the first buyer and the seller about his decision and in the case when he agrees on the price, he proceeds by sending the seller the shipping address and subsequently receives the arrival date of the book. The global type that describes the two buyer and seller example using multicast send of values and labels is defined below:

$$\begin{aligned} G = \ & \mathtt{Buyer1} \to \mathtt{Seller}\colon 1\ \langle \mathit{string}\rangle. \\ & \mathtt{Seller} \to \mathtt{Buyer1}, \mathtt{Buyer2}\colon 1, 2\ \langle \mathit{int}\rangle. \\ & \mathtt{Buyer1} \to \mathtt{Buyer2}\colon 3\ \langle \mathit{int}\rangle. \\ & \mathtt{Buyer2} \to \mathtt{Buyer1}, \mathtt{Seller}\colon 3, 2\ \{\mathit{ok}\colon \mathtt{Buyer2} \to \mathtt{Seller}\colon 2\ \langle \mathit{string}\rangle. \\ & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathtt{Seller} \to \mathtt{Buyer2}\colon 2\ \langle \mathit{Date}\rangle.\mathsf{end} \\ & \qquad\qquad\qquad\qquad\qquad\qquad \mathit{quit}\colon \mathsf{end}\} \end{aligned}$$

Buyer1 and Buyer2 denote respectively the first and second buyer. In the original version, the action of sending the price to both buyers is defined over two distinctive causalities—each sending the price from seller to one of the buyers, while in CSMS that action is naturally modeled in a single causality using multicast send of values, resulting in a minimal global type definition. In the original version, the decision of whether to buy the book by the second buyer is conveyed only to the seller since it would not be possible to express a second label "send" to the first buyer due to restrictions of the type system (related to projection) of Honda et al. (2008b). However, in CSMS the label is sent in multicast to both the first buyer and seller, increasing the expressivity of the protocol.

Below, we define the processes of the protocol, possessing the same benefits as the global type when compared to the processes definition of the original version.

$$
\begin{aligned}
\text{Buyer1} \triangleq\ & \overline{a}_{[2,3]}(x_1, x_2, x_3).\ x_1!\langle\text{“War and Peace”}\rangle; \\
& x_1?(quote);\ x_3!\langle quote\ \text{div}\ 2\rangle; x_3\&\{\mathsf{ok}\colon P_1, \mathsf{quit}\colon \mathbf{0}\} \\
\text{Buyer2} \triangleq\ & a_{[2]}(x_1, x_2, x_3).\ x_2?(quote);\ x_3?(contrib); \\
& \text{if } (quote - contrib \leq 99) \\
& \quad \text{then}\quad x_2, x_3 \oplus \langle\mathsf{ok}\rangle; x_2!\langle address\rangle; x_2?(x); P_2 \\
& \quad \text{else}\quad x_2, x_3 \oplus \langle\mathsf{quit}\rangle; \mathbf{0} \\
\text{Seller} \triangleq\ & a_{[3]}(x_1, x_2, x_3).\ x_1?(title);\quad x_1, x_2!\langle quote\rangle; \\
& x_2\&\{\mathsf{ok} : x_2?(x); x_2!\langle date\rangle; Q, \mathsf{quit} : \mathbf{0}\}
\end{aligned}
$$

where $k_1, k_2!\langle v\rangle$ denotes the action of sending value v in multicast through channels k_1, k_2.

2.4 Higher-order Communication in CSMS

Higher-order communication models the capability of a process to delegate its participation in a session, say a, to another process without letting the other participants know. From a global type perspective, this means that the global type G_a that describes the behaviour of session a must not reflect at any instance the participants that may join the session from delegation. An essential aspect of CSMS is to practically model delegation in processes as a delivery of channels from one process to another, while in the initial system, processes possessed the channels before the delivery has occurred, restricting delegation to bound output.

2.4.1 Modeling in Global Types

Global types describe the interactions between the participants of a session. For example, a session between three participants `A`, `B` and `C`, where `A` and `B` send respectively the author and the title of a book to `C`, is defined as:

$$G_a = \ \mathtt{A} \to \mathtt{C}\colon 1\ \langle string\rangle.\mathtt{B} \to \mathtt{C}\colon 2\ \langle string\rangle.\text{end}.$$

For some reason, participant A leaves that session and delegates its behaviour to another participant D. In the global type syntax, this behaviour is described as:

$$G_b = \quad \mathtt{A} \to \mathtt{D}\colon 1\ \langle 1!\langle string\rangle @\mathtt{A}\rangle.\mathsf{end}$$

where the type of the message $1!\langle string\rangle@\mathtt{A}$ captures the behaviour of participant A in the global type G_a— send the author of the book (string) to C through channel 1. An essential aspect of G_b is that A delegates its behaviour to D without informing the other participants of G_a: $\mathtt{B}, \mathtt{C}$.

The global type G_a describes strictly the global behaviour of the session between participants $\mathtt{A}, \mathtt{B}$ and C and does not reflect —at any instance of the global type— the delegation of A to D. This convention is later used by the type system when typing programs. We emphasise this as it was a source of errors in Honda et al. (2008b) when modeling delegation as explained in the next paragraph.

The global type of the Alice-Bob-Carol example in (Honda et al., 2008b, Section 4.4) violates the above convention and so, ill-types correct processes. In their work, the global type that describes the behaviour between Alice and Carol is defined with the knowledge that Alice will delegate her behaviour to Bob and so in there the global type is defined as:

$$G_b = \mathtt{B} \to \mathtt{C}\colon\ s_1!\langle int\rangle.\mathsf{end}$$

where B stands for Bob and C for Carol. The global type that describes the delegation between Alice and Bob is defined as follows:

$$G_a = \mathtt{A} \to \mathtt{B}\colon\ t_1\langle s_1!\langle int\rangle; \mathsf{end}@\mathtt{B}\rangle.\mathsf{end}$$

Alice's behaviour of initiating the session on b, and subsequently deliver the session channel of b to Bob, is ill-typed by the type system as in the global type G_b, Alice does not appear as a participant. Therefore, the correct global types must define in G_b the behaviour between Alice and Carol and G_a the delegation between Alice and Bob as defined below:

$$G_b = \mathtt{A} \to \mathtt{C}\colon\ s_1!\langle int\rangle.\mathsf{end}$$
$$G_a = \mathtt{A} \to \mathtt{B}\colon\ t_1\langle s_1!\langle int\rangle; \mathsf{end}@\mathtt{A}\rangle$$

2.4.2 Modeling in Processes

Honda et al. (1998, 2008b) model delegation for binary session and multiparty session types by sending all channels of a session. Their proposal has a shortfall that limits its use in practice. That is, higher-order communication is defined as:

$$k!\langle k' \rangle \mid k?\langle k' \rangle$$

where the receiver possesses channel k' before the communication takes place. The calculus of this thesis overcomes this shortfall by adding multipolarity labels to session channels as in Gay and Hole (2005); Yoshida and Vasconcelos (2007). That is, CSMS models the transmission of channels with the receiver not possessing the channels until the communication happens.

The system developed by Honda et al. (1998, 2008b) does not define the term

$$\texttt{throw}\ k[k']; P_1 \mid \texttt{catch}\ k(k'')^1\ \texttt{in}\ P_2 \nrightarrow$$

as semantically correct, where $\mid$ denotes parallel composition of processes. In order to reduce, the receiver must possess the channel k' before the communication take place. This design choice restricts the reduction rules of the calculus to bound output and does not match the implementation of delegation in SJ by Hu et al. (2008)— an extension of the Java programming languages that integrates session programming into an object-oriented framework.

Yoshida and Vasconcelos (2007) describe different extensions of the operational semantics and analyse soundness of the type system to safely support practical higher-order communication for binary session types. They firstly propose to rename the bound channel k'' into k' but that might result in erroneously binding free occurrences of k' into P_2. Another solution suggested was to change the operational semantics rule into

$$\texttt{throw}\ k[k']; P_1 \mid \texttt{catch}\ k(k'')\ \texttt{in}\ P_2 \ \rightarrow\ P_1 \mid P_2[k'/k'']$$

This rule breaks soundness of the type system. Indeed the process

$$\texttt{accept}\ b(k')\ \texttt{accept}\ a(k)\ \texttt{in}\ \texttt{throw}\ k[k'] \mid \texttt{request}\ b(k')\ \texttt{request}\ a(k)\ \texttt{in}\ \texttt{catch}\ k(k'')\ \texttt{in}\ k''?(y)\ \texttt{in}\ k'![1]$$

[1]The terms $\texttt{throw}\ k[k']$ and $\texttt{catch}\ k(k'')$ translate to $k!\langle k' \rangle$ and $k?(k')$ in the syntax of CSMS.

where keywords `request` and `accept` denote respectively the process willing to start and accept a session, is well-typed by Honda et al. (1998) type system but the reduced term

$$k'?(y)\ \texttt{in}\ k'![1]$$

is ill-typed under the same type system since the type of k' captures only one write in the initial term, while in the reduced term it captures one read and one write. Even though, this example might be controversial of its utility in practice, it shows a well formed term of the calculus that breaks soundness of the type system. Hu et al. (2008) in their implementation of SJ reject at runtime the above example as it get stuck in a non-final value and so, blocks progress of the system. Lastly, Yoshida and Vasconcelos (2007) introduce channels as runtime entities; i.e. they are not part of the syntax used by programmers and are generated at initiation time, as in the calculus introduced in this dissertation. The above example is written in their system as follows:

$$\texttt{accept}\ b(y_1)\ \texttt{accept}\ a(y_2)\ \texttt{in}\ \texttt{throw}\ y_2[y_1]\ |\ \texttt{request}\ b(y_1)\ \texttt{request}\ a(y_2)\ \texttt{in}\ \texttt{catch}\ y_2(y_3)\ \texttt{in}\ y_3?(y_4)\ \texttt{in}\ y_1![1].$$

The session channels generated at session initiation time are labelled by a polarity sign (+, -) when substituted into the scope of processes. By convention, the polarity label - is assigned to the channel substituted into the process prefixed by a session request primitive and + into the process prefixed by a session accept primitive. The polarity label is syntax added to channels to extend their notion of belonging to a communication media. In other words, a channel is not only an entity that belongs to a communication but also an entity that belongs to a particular process. The reduced term of the above process

$$\kappa'^{+}?(y)\ \texttt{in}\ \kappa'^{-}![1]$$

is well-typed as k' of one endpoint differs from the other by a polarity sign, resulting in two distinct types.

CSMS uses the mechanism of polarities as Yoshida and Vasconcelos (2007); Gay and Hole (2005) to represent channels at runtime, extending them to multipolarities `1`, ..., `n`. Both those systems use binary polarity (+, -) to specialise channels by participants in binary sessions, while in the system of multiparty sessions, multipolarity is used to specialise the channels for every process. The suffix introduced at each channel reflects the identity of the participant who is using the channel. In contrast to Gay and Hole

(2005), polarities in this system are not part of the user-defined syntax but of the runtime (introduced when channels are created at session initialisation), providing a simpler calculus for the programmers. The reduced term above is defined in CSMS as:

$$\kappa'^{1}?(y) \texttt{ in } \kappa'^{2}![1]$$

where the process with the session request prefix represents the behaviour of participant `1` and the other the behaviour of participant `2`.

Below, we give the implementation of processes in CSMS of the delegation example, described in the previous sub-section. The reduction steps are provided in Section 2.6.2.

$A \triangleq \overline{a}_{[2,\,3]}\,(x_1, x_2).\overline{b}_{[2]}\,(y_1).y_1!\langle x_1, x_2\rangle;$
$B \triangleq a_{[2]}\,(x_1, x_2).x_2!\langle \text{“}\textit{The computer and the brain}\text{”}\rangle;$
$C \triangleq a_{[3]}\,(x_1, x_2).x_1?(y'); x_2?(y'');$
$D \triangleq b_{[2]}\,(y_1).y_1?(y_2, y_3); y_2!\langle \text{“}\textit{John von Neumann}\text{”}\rangle;$

where participant `A` delegates ($y_1!\langle x_1, x_2\rangle;$) its behaviour —send the author of the book— to participant `D`, and participant `B` sends the title of the book to participant `C`.

2.5 Linearity in CSMS

The context of study for multiparty sessions by Honda et al. (2008b) and CSMS is not limited to communication media that are not shared between different peers such as internet sockets which are not shared between different machines. It is extended to communication media that are shared across different peers as memory is shared by several cores and process units to communicate with each other in recent computer architectures. In this section, we introduce a linearity property to check a global type whether the use of a channel in two different communications may break the causalities specified in the global description at runtime. This is because global types on their own do not guarantee that communications at run-time will be the same as those specified in the global behaviour.

For example, two processes receive a different message from a third process. The global type that describes this behaviour must define two causalities on two different channels so that the message sent to one process is not received by the other. However, the global types syntax allows programmers to write the fore-mentioned behaviour using only one channel, for example,

$$\mathtt{A} \to \mathtt{B} : k\langle int\rangle.\mathtt{A} \to \mathtt{C} : k\langle string\rangle$$

and the type system well-types the following program:

$$k?\langle y\rangle \mid k?\langle y\rangle \mid k!\langle 3\rangle; k!\langle \text{“}Blue\text{”}\rangle$$

where the first process represents the behaviour of A, the second process the behaviour of B and the third process the behaviour of C. For simplicity, we have omitted the session initiation prefix from the definition of the processes. At runtime, the second process can receive the first message sent from the third process, breaking the causalities of the global type. Ambiguity of interactions introduced by an incorrect use of channels can be present in processes and the consequences of running these sessions are the same as the ones obtained by an incorrect implementation of a protocol—breaking the logic of programs.

In the Addition example described in Section 2.1, in the $false$ branch, *addition* uses different channels to send the values of the operands to *successor* ($x_2!\langle y_1\rangle$) and *predecessor* ($x_3!\langle y_2\rangle$) to not break the causalities of the global type and so the logic of the program, since there is no guarantee that the receptions will be in a fixed order. That is, if we were to use x_2 for both communications, the message of the first operand could be received by *predecessor* rather than *successor* and so, return a wrong result to the client. The problem becomes visible after the fifth step of reduction, given in Section 2.6.2.

A precise analysis of channel dependencies between two synchronous communications— input-input, input-output, output-input, output-output, output-output and input-input, input-output and output-input— is provided in Figure 2.1. The letters A and S denote respectively the asynchronous and synchronous calculus. All six cases are considered in CSMS unlike the asynchronous calculus of Honda et al. (2008b) where the output-input (OI) and output-output (OO) cases are not considered. In asynchronous communications, the output-input case is not considered since the reception of the message from $\mathtt{P}_2$ can occur before $\mathtt{P}_1$ receives the message sent. Similarly for the output-output case, the second message sent can be received before the first one. As shown in the figure, the causalities in the II, IO, OI and OO cases can be broken by processes when defined over the same channel. Indeed, as demonstrated by the processes (the third line), in the (II) case: p can receive the first message from $\mathtt{p}_2$ rather than $\mathtt{p}_1$, in the (IO) case: the message sent by $\mathtt{p}_1$ can be received by $\mathtt{p}_2$, in the (OI) case: the message sent by $\mathtt{p}_2$ can be received by $\mathtt{p}_1$ and in the (OO) case: $\mathtt{p}_2$ can receive the first message rather than $\mathtt{p}_1$. This analysis leads to the definition of a linearity property describing when it is safe, in terms of non-ambiguity, for

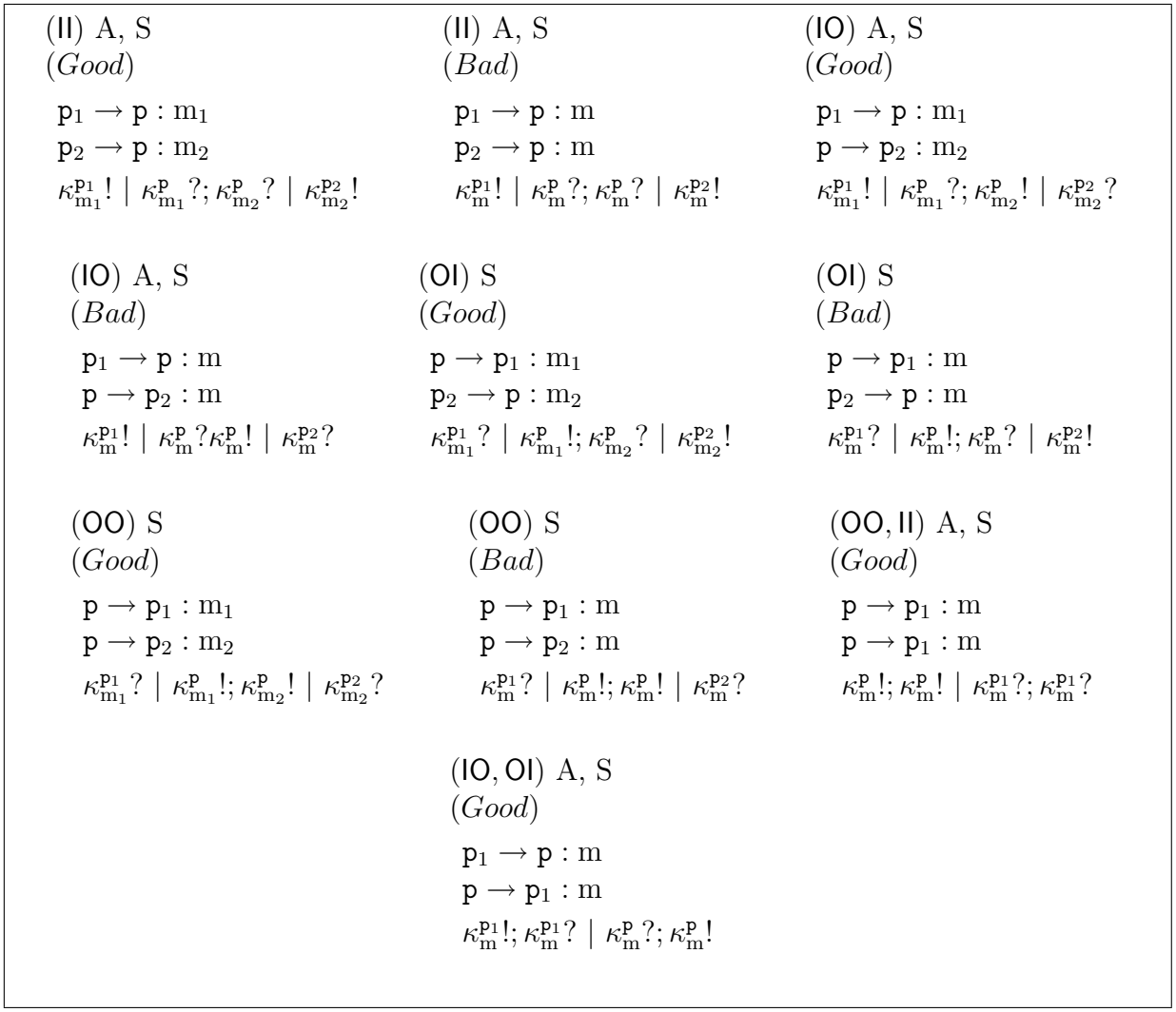


Figure 2.1: CSMS: Causality analysis.

two communications to use the same channel.

Linearity checks two causalities that have the same channel for *input* and *output* dependencies. The input dependency defines a relation between the first communication and the action that precedes the receiving action of the second communication as illustrated in the diagram in Figure 2.2. Input dependency assures that the "receive" of the second communication does not interfere with the receive of the first communication and so, it is preceded by an action that takes place only after the first communication has occurred. The diagram simplifies the idea of input dependency in a scenario where both causalities $P_1 \to P_2 : \kappa_1$ and $P_n \to P_{n+1} : \kappa_1$ are defined over the same channel κ_1. Process P_{n+1} is prefixed by an action (receiving or sending) on a channel κ_l and that action takes place only after the communication between P_1 and P_2 has occurred. The order is given

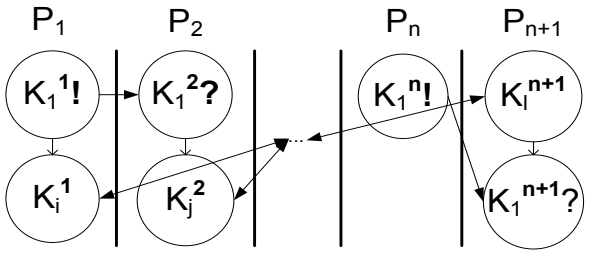


Figure 2.2: CSMS: Input dependency chain.

graphically through the arrows, denoting communications, starting either after the sending action on P_1 or after the receiving action on P_2. The formal dependency relation between $P_1 \rightarrow P_2 : \kappa_1$ and the causality on κ_l of P_{n+1} is given in Section 2.6.3.

The output dependency defines a relation between the first communication and the action that precedes the sending action of the second communication as illustrated in the diagram in Figure 2.3. The property that output dependency assures is that the "send" of the second communication does not interfere with the "send" of the first communication and so, it is preceded by an action that takes place only after the first communication has occurred. The diagram simplifies the idea of output dependency in the same scenario as above. Process P_n is prefixed by an action (receiving or sending) on some channel κ_l and

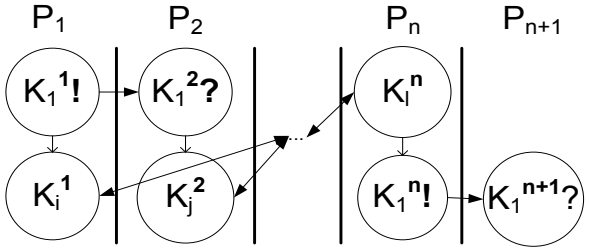


Figure 2.3: CSMS: Output dependency chain.

that action takes place only after the communication between P_1 and P_2 has occurred. The order is given graphically through the arrows, starting either after the sending action on P_1 or after the receiving action on P_2. The dependency relation between $P_1 \rightarrow P_2 : \kappa_1$ and the causality on κ_l of P_n is given in Section 2.6.3.

When both input and output dependencies exist between two different communications that are defined over the same channel then there is no interference between the "send" and "receive" operations of the second communication with the first one, resulting in a

non-ambiguous use of the same channel. The initial work on multiparty session types lacks from such intuition, making it difficult to understand the formal definition of linearity with unnecessary conditions as we shall see in Section 2.6.3.

Buyer-Seller Example with Three Channels. Programming sessions with the minimum number of channels without the support of any automatic tool is difficult. Below, we present a version of the example given in Section 2.3 for the asynchronous calculus with queue of Honda et al. (2008b), using only three channels rather than four as in the original version.

$$
\begin{array}{rcl}
\text{Buyer1} & \triangleq & \overline{a}_{[2,3]}\,(b_1, b_2, s).\;\; s!\langle\text{“War and Peace”}\rangle; \\
& & \qquad b_1?(\mathit{quote});\;\; s!\langle \mathit{quote}\;\text{ div }\;2\rangle; P_1 \\
\text{Buyer2} & \triangleq & a_{[2]}\,(b_1, b_2, s).\;\; b_2?(\mathit{quote});\;\; s?(\mathit{contrib}); \\
& & \qquad \text{if } (\mathit{quote} - \mathit{contrib} \;\leq\; 99) \\
& & \qquad\quad \text{then} \quad b_2 \oplus \langle \mathsf{ok} \rangle; b_2!\langle \mathit{address} \rangle; b_1?(x); P_2 \\
& & \qquad\quad \text{else} \quad b_2 \oplus \langle \mathsf{quit} \rangle; \mathbf{0} \\
\text{Seller} & \triangleq & a_{[3]}\,(b_1, b_2, s).\;\; s?(\mathit{title});\quad b_1, b_2!\langle \mathit{quote} \rangle; \\
& & \qquad b_2 \rhd \{\mathsf{ok}\colon\, b_2?(x); b_1!\langle \mathit{date} \rangle; Q\,,\quad \mathsf{quit}\colon\, \mathbf{0}\}
\end{array}
$$

One can easily verify that there are no race conditions on the channels at any communication in this program and the following note may help the sceptical reader: the branching of Seller on channel b_2 does not interfere with the “receive” of the quote on the same channel in the second buyer by definition of the operational semantics; i.e., there are two distinct rules that define reception of values and labels. This example shows the need for automated tools that generate the minimum number of channels in a session without race conditions. A minimum number of channels is important as it reduces the cost of the delegation operation; recalling the definition of the latter: delivery of all session channels. A smart use of linearity is to define the channels of global types safely and minimally, instead of using the linearity property to check whether the programmer has used safely the same channel in more than one communication.

2.6 Formal Model

Our system is modelled on that of Honda et al. (2008b) as the first work on multiparty

sessions that offers a practical use in programming languages. CSMS minimises the definition of the multiparty calculus of the previous work, representing a primitive value send and a session channel send through a single primitive, following the idea firstly proposed for binary session types in Gay and Hole (2005). In contrast to the previous work, this system does not support multithreading as the challenges of that topic go beyond the study of enriching the multiparty session theory with synchronous communications, multicasting, practical higher-order communication and other features discussed in this chapter. The model is based on a small-step operational semantics, which allows the use of standard proof techniques.

Multicasting of values and labels may form global types that correspond to meaningless and erroneous sessions. We introduce a well-formedness system that ensures a global type that for each receiver (of a value or label) corresponds a channel that performs the delivery and that a behaviour is sent (as delegation) in multicast if the sub-behaviours of the decomposed behaviour are independent (sub-behaviours which order of evaluation does not affect the logic of the program). A linearity property is defined to check for race conditions the channels of a global type and so, later type-check the program. Our definition of linearity is modelled after that of the previous work but improves it in a more simple and declarative version.

The static type system of multiparty session typing allows lightweight type annotations and efficient typing strategy of simply global types; i.e., programmers first define the global type of the intended interaction and then define each process of it, and processes are validated through projection of the global type onto the principals by type-checking. We extend the type system of the initial work to support multicast send of values, including higher-order communication, and labels.

2.6.1 Syntax

Figure 2.4 gives the syntax of global session types or global types as we refer to them throughout this thesis. The metavariable G (as well as with subscript and suffix) stands for global types; U stands for message types, including end-point types T (see Figure 2.9) and value types V. The metavariable $\mathtt{p}$ (as well as with suffix and subscript) ranges over participants; n (as well as m with suffix and subscript) ranges over channels of natural numbers; (as well as with subscript) ranges over labels; i ranges over indexes of natural numbers and I (as well as J) ranges over sets of natural numbers; $\mathbf{t}$ ranges over type

$G ::=$		Global types:	$\mid \mu\mathbf{t}.G$	Recursion
$\mid$	$\mathtt{p} \to \tilde{\mathtt{p}}' \colon \tilde{\mathtt{n}}\,\langle \tilde{U} \rangle.G$	Message	$\mid \mathbf{t}$	Type variable
$\mid$	$\mathtt{p} \to \tilde{\mathtt{p}}' \colon \tilde{\mathtt{n}}\,\{l_i : G_i\}_{i \in I}$	Branching	$\mid \mathsf{end}$	End
$U ::= V \mid T@\mathtt{p}$	Message types	$V ::= \mathsf{bool} \mid \mathsf{nat} \mid .. \mid \langle G \rangle$		Value types

Figure 2.4: CSMS: Syntax of global session types.

variables. $\tilde{\mathtt{p}}$, $\tilde{\mathtt{n}}$ represent respectively a sequence of participants and channels that does not contain duplicated elements. $\tilde{U}$ represents a list of types U.

Global types The construct $\mathtt{p} \to \tilde{\mathtt{p}}' \colon \tilde{\mathtt{n}}\,\langle \tilde{U} \rangle.G$ captures a message exchange of type U between participants $\mathtt{p}$ and $\tilde{\mathtt{p}}'$, respectively sender and receivers, through channels $\tilde{\mathtt{n}}$. The right hand side of the causality ($\to$) is defined as a list of participants to support multicast send of values, representing single or multiple receivers. The number of receivers must be the same as one of the channels, i.e. the cardinality of lists $\tilde{\mathtt{p}}'$ and $\tilde{\mathtt{n}}$ must be the same, and channels must be different (These constraints are enforced by the well-formedness rules in Figure 2.8). Delegation is described as a message exchange of type end-point T where T captures the behaviour of the channels sent. A behaviour delegated in multicast must be divided into independent pieces and the number of pieces must be the same as the number of channels and receivers (These constraints are ensured by the well-formedness rules in Figure 2.8). The exchange of a shared name between two or more participants is described as a message exchange of type $\langle G \rangle$.

Notation 2.6.1. The notation $T@\mathtt{p}$ present in the definition of U denotes an end-point type T assigned to participant $\mathtt{p}$.

The construct $\mathtt{p} \to \tilde{\mathtt{p}}' \colon \tilde{\mathtt{n}}\,\{l_i : G_i\}_{i \in I}$ captures branching of a conversation over a number of labels. Participant $\mathtt{p}$ internally chooses one of the labels l_i enumerated by I and then sends it to participants $\tilde{\mathtt{p}}'$ and the conversation follows G_i. Similarly to the message construct, the right hand side of the causality is defined as a list of participants to support multicast send of labels. Also, the number of receivers must be the same as one of the channels and channels must be different (These constraints are enforced by the well-formedness rules in Figure 2.8).

Infinite behaviour is represented by recursively defined global types $\mu\mathbf{t}.G$. end signifies the end of a conversation and is used as a base type to build more complex global types. A type U ranges over primitive types (bool, nat, ...), global types ($\langle G \rangle$), and end-point types (T) (see Figure 2.9).

$P ::=$	Processes:		
$\mid \bar{u}[2..\mathrm{n}](\tilde{x}).P$	Multicast request	$\mid$ if e then P else Q	Conditional
$\mid u[\mathtt{p}](\tilde{x}).P$	Accept	$\mid P \mid Q$	Parallel
$\mid \tilde{k}!\langle\tilde{e}\rangle; P$	Value sending	$\mid \mathbf{0}$	Inaction
$\mid k?\langle\tilde{x}\rangle; P$	Value reception	$\mid (\nu w)P$	Hiding
$\mid \tilde{k} \oplus \langle l\rangle; P$	Selection	$\mid$ def D in P	Recursion
$\mid k\&\{l_i : P_i\}_{i\in I}$	Branching	$\mid X\langle\tilde{e}\rangle$	Process call

$u ::= x \mid a$ Identifiers $\quad e ::= x \mid v \mid e \text{ and } e' \mid \text{not } e \mid \cdots$ Expressions

$k ::= x \mid \kappa^{\mathrm{n}}$ Session channels $\quad v ::= a \mid \kappa^{\mathrm{n}} \mid \mathrm{n} \mid \mathbf{true} \mid \mathbf{false}$ Values

$\mathrm{D} ::= X_1(\tilde{x}_1) = P_1 \text{ and } \cdots \text{ and } X_{\mathrm{n}}(\tilde{x}_{\mathrm{n}}) = P_{\mathrm{n}}$ Recursive declaration

Figure 2.5: CSMS: User-defined and runtime syntax of processes.

Processes Figure 2.5 introduces the syntax of processes. The metavariable P (as well as Q and with subscript) stands for processes; u stands for identifiers of a session; k stands for session channels; e stands for expressions, including natural and Boolean expressions; v stands for values; D stands for recursive declarations. The variable $\mathtt{p}$ (as well as $\mathtt{q}$ and with subscript) is a metavariable ranging over participants; (as well as with subscript) ranges over labels; i (as well as j) ranges over indexes of natural numbers; I (as well as J) ranges over sets of natural numbers; w ranges over identifiers name and session channel values; X (as well as with subscript) ranges over process variables; a ranges over shared names; κ^{n} ranges over session channel values, where n ranges over natural numbers; x (as well as y, z and with subscript) ranges over variables of the calculus. $\tilde{k}$ represents a sequence of channels that does not contain duplicated elements.

The process with the session request prefix $\bar{u}[2..\mathrm{n}](\tilde{x}).P$ describes the behaviour of participant 1 willing to initiate a session with participants 2, ..., n. A session is established between a minimum of two participants, syntactically enforced by imposing the lower bound of n to 2. In the prefix, the session identifier u serves as a public point of access for all the participants to dynamically set the session channels (place held by $\tilde{x}$) that perform the actions defined in P. The shared name is similar to a lock that serves as a point of access for all persons who enter a room/building. The relation between shared names and participant identifiers is similar as the lock and keys. Reciprocally to the process that sends a request for session initiation, the processes $u[\mathtt{p}](\tilde{x}).P$, where $\mathtt{p}$ ranges over 2, ..., n, accept that request. The other constructs of the prefix are similar to the Request process prefix.

The sending construct $\tilde{k}!\langle\tilde{e}\rangle; P$ describes the action of sending single or multiple values

to single or multiple participants. The list of session channels $\tilde{k}$ may include all the channels of a session, representing a multicast send to all the group of participants, or a subset of them, representing a multicast send to a particular subgroup. Higher-order communication is achieved by sending shared names or session channels. Reciprocally, $k?\langle\tilde{x}\rangle;P$ describes the receiving action of single or multiple values place held by $\tilde{x}$ over the channel k.

Branching over paths of a conversation is defined with labels. The selection construct $\tilde{k} \oplus \langle l_i \rangle; P$ chooses one of the labels l_i enumerated by I and sends it to single or multiple participants. Reciprocally, the branching construct $k\&\{l_i : P_i\}_{i \in I}$ splits the behaviour of a participant over the labels l_i.

The conditional has the same definition as in imperative languages. The parallel construct is standard, composing in parallel the behaviour of two processes. It defines computation in the calculus in a similar way as application defines computation in the λ-calculus. **0** describes the end of a behaviour. Hiding is standard, restricting the scope of actions of shared names (a) and session channels (κ) only to P. This adds privacy to channels to disallow messages (send by those channels) to escape the scope of processes. Similarly, a private language established between persons does not allow other persons to understand the words spoken by the former participants. This is a common property found in many communication abstraction such as communicators in MPI.

The construct $\mathsf{def}\ D\ \mathsf{in}\ P$ defines infinite behaviour in the calculus through recursive functions. $X_1(\tilde{x}_1) = P_1\ \mathsf{and}\ \cdots\ \mathsf{and}\ X_{\mathrm{n}}(\tilde{x}_{\mathrm{n}}) = P_{\mathrm{n}}$ shows how to define recursive function over single or multiple parameters in a similar way to ML by Milner et al. (1997). These functions are instantiated through the $X\langle\tilde{e}\rangle$ construct, typically present in the scope of P.

Identifiers of a session include variables and shared names. Session channels consist of variables and session channel values labelled by a natural number— the only construct that is part only of the runtime syntax. The label identifies a participant inside a session just as an IP address identifies a computer in the Internet or as a rank (natural number) identifies a process inside a communicator in MPI. Expressions include shared names, session channel values, natural numbers, Booleans and other primitive values.

The association of the parallel operator $\mid$ is the weakest over all the prefixes and operators (ν, $\mathsf{def}\ D\ \mathsf{in}\ P$).

Definition 2.6.2. The set of free names of a process P, written $fn(P)$, is defined as follows:

$$fn(\bar{u}[2..\mathrm{n}](\tilde{x}).P) = fn(u) \bigcup fn(P) \quad fn(\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q) = fn(P) \bigcup fn(Q)$$
$$fn(u[\mathsf{p}](\tilde{x}).P) = fn(u) \bigcup fn(P) \quad P \mid Q = fn(P) \bigcup fn(Q)$$

$$fn(\tilde{k}!\langle\tilde{e}\rangle; P) = fn(\tilde{e}) \bigcup fn(P) \quad fn(\mathbf{0}) = \emptyset$$
$$fn(k?\langle\tilde{x}\rangle; P) = fn(P) \quad fn((\nu w)P) = fn(P) \setminus fn(w)$$
$$fn(\tilde{k} \oplus \langle l\rangle; P) = fn(P) \quad fn(\mathsf{def}\ D\ \mathsf{in}\ P) = fn(D) \bigcup fn(P)$$
$$fn(k\&\{l_i : P_i\}_{i\in I}) = \bigcup_{i\in I} fn(P_i) \quad fn(X\langle\tilde{e}\rangle) = fn(\tilde{e})$$

$$fn(\mathrm{x}) = \emptyset \quad fn(a) = \{a\} \quad fn(\kappa_{\mathrm{n}}) = \emptyset \quad fn(\mathrm{n}) = \emptyset$$
$$fn(X_1(\tilde{x}_1) = P_1\ \mathsf{and}\ \cdots\ \mathsf{and}\ X_{\mathrm{n}}(\tilde{x}_{\mathrm{n}}) = P_{\mathrm{n}}\ ) = \bigcup_{i\in\{1..\mathrm{n}\}} fn(P_i)$$

2.6.2 Operational Semantics

Figure 2.6 gives the operational semantics of CSMS in a small step style via the reduction relation $\longrightarrow$ where the state of the machine is defined only by terms of the calculus, written "$P \longrightarrow P'$", read "process P reduces to process P' in one step". The relation is defined over six computation rules (axioms) and four congruence rules (inference rules).

The rule [R-Link] invokes a session between n peers by generating fresh session channels $\tilde{\kappa}$ and places them into the processes scope through substitution. The substitution rule is the same as in the π-calculus, following the same scoping constraints when substituting variables. The channels of each process are characterised by the participant identifier to define the identity of each participant within the session. The n-party synchronisation captures a handshake process, as in telecommunications, of negotiation that dynamically sets the channels between n participants before normal communication between processes P_1, ..., P_{n} starts. The session channels created are private, providing an identity to the session. Below we give the notation and definition of substitution:

Notation 2.6.3. Substitution is a function $\sigma = \{v_1, ..., v_n/x_1, ..., x_n\}$ that replaces named variables by values such as $x_i\sigma = v_i$ for every $i \in [1..n]$ and $x\sigma = x$ if $x \notin \{x_1, ..., x_n\}$.

Definition 2.6.4 (Substitution)**.** The process $P\sigma$ obtained by applying σ to P is defined as follows:

$$\begin{array}{lcllcl}
(\bar{u}[2..\mathrm{n}](\tilde{x}).P)\ \sigma & = & \bar{u\sigma}[2..\mathrm{n}](\tilde{x}).P\sigma & (\mathbf{0})\sigma & = & \mathbf{0} \\
(u[\mathsf{p}](\tilde{x}).P)\ \sigma & = & u\sigma[\mathsf{p}](\tilde{x}).P\sigma & ((\nu w)P)\ \sigma & = & (\nu w)P\sigma \\
(\tilde{k}!\langle\tilde{e}\rangle; P)\sigma & = & \tilde{k}\sigma!\langle\tilde{e}\sigma\rangle; P\sigma & (\mathsf{def}\ D\ \mathsf{in}\ P)\sigma & = & \mathsf{def}\ D\sigma\ \mathsf{in}\ P\sigma \\
(k?\langle\tilde{x}\rangle; P)\sigma & = & k\sigma?\langle\tilde{x}\rangle; P\sigma & (X\langle\tilde{e}\rangle)\sigma & = & X\langle\tilde{e}\sigma\rangle \\
(\tilde{k} \oplus \langle l\rangle; P)\sigma & = & \tilde{k}\sigma \oplus \langle l\rangle; P\sigma & (k\&\{l_i : P_i\}_{i\in I})\sigma & = & k\sigma\&\{l_i : P_i\sigma\}_{i\in I}
\end{array}$$

$$\bar{a}[2..\mathrm{n}](\tilde{x}).P_1 \mid a[2](\tilde{x}).P_2 \mid \cdots \mid a[\mathrm{n}](\tilde{x}).P_\mathrm{n} \longrightarrow (\nu\tilde{\kappa})(P_1\{\tilde{\kappa}^1/\tilde{x}\} \mid P_2\{\tilde{\kappa}^2/\tilde{x}\} \mid \cdots \mid P_\mathrm{n}\{\tilde{\kappa}^\mathrm{n}/\tilde{x}\}) \quad \text{[R-Link]}$$

$$\tilde{\kappa}^\mathtt{p}!\langle\tilde{v}\rangle; P \mid \kappa_{\mathrm{m}_1}^{\mathtt{p}_1}?\langle\tilde{x}\rangle; P_1 \mid \cdots \mid \kappa_{\mathrm{m}_\mathrm{n}}^{\mathtt{p}_\mathrm{n}}?\langle\tilde{x}\rangle; P_\mathrm{n} \longrightarrow P \mid P_1\{\tilde{v}/\tilde{x}\} \mid \cdots \mid P_\mathrm{n}\{\tilde{v}/\tilde{x}\} \quad (\{\kappa_{\mathrm{m}_1}, ..., \kappa_{\mathrm{m}_\mathrm{n}}\} = \{\tilde{\kappa}\}) \quad \text{[R-Multicast]}$$

$$\tilde{\kappa}^\mathtt{p} \oplus \langle l_i\rangle; P \mid \kappa_{\mathrm{m}_1}^{\mathtt{p}_1}\&\{l_j : P_{1j}\}_{j\in I} \mid \cdots \mid \kappa_{\mathrm{m}_\mathrm{n}}^{\mathtt{p}_\mathrm{n}}\&\{l_j : P_{\mathrm{n}j}\}_{j\in I} \longrightarrow P \mid P_{1i} \mid \cdots \mid P_{\mathrm{n}i} \quad (\{\kappa_{\mathrm{m}_1}, ..., \kappa_{\mathrm{m}_\mathrm{n}}\} = \{\tilde{\kappa}\}, i \in I) \quad \text{[R-Label]}$$

$$\mathsf{if}\ \mathbf{true}\ \mathsf{then}\ P\ \mathsf{else}\ Q \longrightarrow P \quad \text{[R-IfT]}$$

$$\mathsf{if}\ \mathbf{false}\ \mathsf{then}\ P\ \mathsf{else}\ Q \longrightarrow Q \quad \text{[R-IfF]}$$

$$\mathsf{def}\ D\ \mathsf{in}\ (X\langle\tilde{v}\rangle \mid Q) \longrightarrow \mathsf{def}\ D\ \mathsf{in}\ (P\{\tilde{v}/\tilde{x}\} \mid Q) \quad (X(\tilde{x}) = P \in D) \quad \text{[R-ProcCall]}$$

$$P \longrightarrow P' \Rightarrow (\nu w)P \longrightarrow (\nu w)P' \quad \text{[R-Scop]}$$

$$P \longrightarrow P' \Rightarrow P \mid Q \longrightarrow P' \mid Q \quad \text{[R-Par]}$$

$$P \longrightarrow P' \Rightarrow \mathsf{def}\ D\ \mathsf{in}\ P \longrightarrow \mathsf{def}\ D\ \mathsf{in}\ P' \quad \text{[R-CProcCall]}$$

$$P \equiv P' \ \text{ and } \ P' \longrightarrow Q' \ \text{ and } \ Q' \equiv Q \ \Rightarrow \ P \longrightarrow Q \quad \text{[R-StructC]}$$

$$e \downarrow v \quad \Rightarrow \quad \mathcal{R}[e] \longrightarrow \mathcal{R}[v] \quad \text{[ContextE]}$$

Figure 2.6: CSMS: Operational semantics.

$$(P \mid Q)\sigma = P\sigma \mid Q\sigma \qquad \begin{array}{lcl}(e\ \mathsf{and}\ e')\sigma & = & e\ \mathsf{and}\ \sigma e'\sigma \\ (\mathsf{not}\ e)\sigma & = & \mathsf{not}\ e\sigma\end{array}$$

$$\begin{array}{lcl}(\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q)\sigma & = & \mathsf{if}\ e\sigma\ \mathsf{then}\ P\sigma\ \mathsf{else}\ Q\sigma \\ (X_1(\tilde{x}_1) = P_1\ \mathsf{and}\ \cdots\ \mathsf{and}\ X_\mathrm{n}(\tilde{x}_\mathrm{n}) = P_\mathrm{n})\sigma & = & X_1(\tilde{x}_1) = P_1\sigma ... X_n(\tilde{x}_n) = P_n\sigma\end{array}$$

A sender sends single or multiple values to single or multiple receivers as defined in rule [R-Multicast]. The result is the next term P from the sender process and the next terms of receivers P_1, ..., P_n with values $\tilde{v}$ substituted in their scope. Communication is defined by the same set of channels from the sender and receivers side. The relation $\downarrow$ evaluates an expression e to a value v and a value v to itself. All the channels that denote a session must be sent when delegating a behaviour (in multicast or not), independently of the channels used to perform that behaviour (This constraint is enforced by the type system in Figure 2.10).

A process internally chooses one of the labels l_i enumerated by I and then sends it to the other processes prefixed by the branching construct though rule [R-Label]. The result is the next term P from the sender ("selector") process and paths $P_{1i}, ..., P_{\mathrm{n}i}$ from the "branching" processes. Similarly to rule [R-Multicast], communication is defined by

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

$$(\nu w)P \mid \mathrm{Q} \equiv (\nu w)(P \mid Q) \quad \text{if } w \notin fn(\mathrm{Q})$$

$$(\nu w w')P \equiv (\nu w' w)P \qquad (\nu w)\mathbf{0} \equiv \mathbf{0} \qquad \mathsf{def}\ D\ \mathsf{in}\ \mathbf{0} \equiv \mathbf{0}$$

$$\mathsf{def}\ D\ \mathsf{in}\ (\nu w)P \equiv (\nu w)\mathsf{def}\ D\ \mathsf{in}\ P \quad \text{if } w \notin fn(D)$$

$$(\mathsf{def}\ D\ \mathsf{in}\ \mathrm{P}) \mid \mathrm{Q} \equiv \mathsf{def}\ D\ \mathsf{in}\ (\mathrm{P} \mid \mathrm{Q}) \quad \text{if } \mathsf{dv}\ (D) \cap \mathrm{fpv}(Q) = \emptyset$$

$$\mathsf{def}\ D\ \mathsf{in}\ (\mathsf{def}\ D'\ \mathsf{in}\ \mathrm{P}) \equiv \mathsf{def}\ D, D'\ \mathsf{in}\ \mathrm{P} \quad \text{if } \mathsf{dv}\ (D) \cap \mathsf{dv}\ (D') = \emptyset$$

$$(\nu k)(\nu k')P \equiv (\nu k, k')P$$

Figure 2.7: CSMS: Structural congruence.

the same set of channels from the sender and receivers side. This rule checks if the label selected by the process with the selection prefix is contained in the labels of the branching processes.

Rule [R-IfT] and [R-IfF] action the evaluation of e; if e evaluates to **true** then rule [R-IfT] is applied otherwise rule [R-IfF]. Rule [R-ProcCall] invokes the process P identified by X with arguments $\tilde{v}$ in the context, resulting in the process P with parameters $\tilde{x}$ replaced by arguments $\tilde{v}$. The process Q represents other calls to processes present in D, typically used to represent multiple recursive behaviour of different processes that are part of the same session.

Rule [R-Scop] actions the reduction of the process inside the scope of the ν operator. Rule [R-Par] defines computation in a parallel composed process, reducing first the subprocesses. Similarly, rule [R-CProcCall] reduces first the subprocess P in the scope of the recursive process. Rule [R-StructC] states that the reduction relation is defined on structural congruent terms. Structural congruence $\equiv$ is the smallest congruence on processes that satisfies the axioms shown in Figure 2.7. As a congruence relation, $\equiv$ defines all the processes that have the same observable behaviour but different syntax. In other words, given two structurally congruent processes they will behave the same in every possible context. Therefore, it is of interest to build a reduction relation that is also defined on structurally congruent terms. fpv() and $\mathsf{dv}()$ denote respectively the set of free process variables and process variables $\{X_i\}_{i \in I}$ in $\{X_i(\tilde{x}) = P_i\}_{i \in I}$. Rule [ContextE] reduces the expressions, denoting messages inside the scope of sending actions, conditionals, process call, to values.

Definition 2.6.5. The runtime term $\mathcal{R}$ with the hole "[]" defines the evaluation contexts of expressions

$\mathcal{R}$ ::=		Expressions evaluation contexts
	$[_]$	Hole
|	$\mathcal{R}$ op e	Expression Left
|	v op $\mathcal{R}$	Expression Left
|	$k!\langle\mathcal{R}\rangle$	Send
|	if $\mathcal{R}$ then P else Q	Conditional
|	$X\langle\tilde{v}\mathcal{R}\tilde{e}\rangle$	Process Call

Reduction steps of the Addition example Below we present the reduction steps of the Addition example described in Section 2.1 where processes Q, R and S are equal by definition to:

$$
\begin{aligned}
Q \triangleq\ & \mathsf{if}(y_2 = 0)\ \mathsf{then}\ x_2, x_3 \oplus\ true; x_1 \oplus\ answer; x_1!\langle y_1\rangle; P \\
& \mathsf{else}\ \ x_2, x_3 \oplus false; x_2!\langle y_1\rangle; x_3!\langle y_2\rangle; x_2?\langle y_1\rangle; x_3?\langle y_2\rangle; X_1\langle y_1, y_2, x_1, x_2, x_3\rangle \\
R \triangleq\ & x_2\ \&\ \{true\colon \mathbf{0}, false\colon x_2?\langle y\rangle; x_2!\langle y+1\rangle; X_2\langle x_2\rangle\} \\
S \triangleq\ & x_3\ \&\ \{true\colon \mathbf{0}, false\colon x_3?\langle y\rangle; x_3!\langle y-1\rangle; X_3\langle x_3\rangle\}
\end{aligned}
$$

The four processes composed in parallel are structurally congruent to the recursive process that contains the union of the recursive functions and the parallel composition of all process, according to the rules in Figure 2.7. The first step initiates the session between *client*, *addition*, *successor* and *predecessor*. We have chosen to present as second step, the delivery of the two operands between *client* and *addition* and not the process call of *successor* and *predecessor* for presentation reasons. Subsequently, in the next three steps, the behaviours of *addition*, *predecessor* and *successor* are invoked, resulting in the substitution of the session channels and operands in the behaviour scope. In the next steps, *addition* internally chooses the second (**false**) branch of the conditional and so, sends it in multicast to *successor* and *predecessor*. This is followed by the delivery of each operand to *successor* and *predecessor* through two distinct channels so that the message sent for *successor* is not received by *predecessor*. The result is then returned by the latter to *addition*. The last four steps are repeated until the addition of the two operands has been accomplished, and lastly the result is sent to *client* by *addition*. For presentation reasons, we have omitted the said steps.

$$client \mid addition \mid successor \mid predecessor \longrightarrow^{[R-StructC],[R-Link]}$$

$$\mathsf{def}\ X_1\langle y_1, y_2, x_1, x_2, x_3\rangle = Q, X_2\langle x_2\rangle = R, X_3\langle x_3\rangle = S\ \mathsf{in}$$

$$
\begin{array}{ll}
(\nu\kappa_1, \kappa_2, \kappa_3) & (\kappa_1^1!\langle 5, 4\rangle; \kappa_1^1\&\{answer : \kappa_1^1?\langle y_1\rangle; 0 \\
& \mid \kappa_1^2?\langle y_1, y_2\rangle; X_1\langle y_1, y_2, \kappa_1^2, \kappa_2^2, \kappa_3^2\rangle \\
& \mid X_2\langle\kappa_2^3\rangle \\
& \mid X_3\langle\kappa_3^4\rangle) \longrightarrow^{[R-CProcCall],[R-Scop],[R-Multicast]}
\end{array}
$$

$$
\begin{array}{l}
\mathsf{def}\ X_1\langle y_1, y_2, x_1, x_2, x_3\rangle = Q, X_2\langle x_2\rangle = R, X_3\langle x_3\rangle = S\ \mathsf{in} \\
\begin{array}{ll}
(\nu\kappa_1, \kappa_2, \kappa_3) & (\kappa_1^1\&\{answer : \kappa_1^1?\langle y_1\rangle; 0 \\
& \mid X_1\langle 5, 4, \kappa_1^2, \kappa_2^2, \kappa_3^2\rangle \\
& \mid X_2\langle\kappa_2^3\rangle \\
& \mid X_3\langle\kappa_3^4\rangle) \longrightarrow^{[R-CProcCall],[R-Scop],[R-ProcCall],[R-ProcCall],[R-ProcCall]}
\end{array}
\end{array}
$$

$$
\begin{array}{l}
\mathsf{def}\ X_1\langle y_1, y_2, x_1, x_2, x_3\rangle = Q, X_2\langle x_2\rangle = R, X_3\langle x_3\rangle = S\ \mathsf{in} \\
\begin{array}{ll}
(\nu\kappa_1, \kappa_2, \kappa_3) & (\kappa_1^1\&\{answer : \kappa_1^1?\langle y_1\rangle; 0 \\
& \mid \mathsf{if}\ (4 = 0)\ \mathsf{then}\ \kappa_2^2, \kappa_3^2 \oplus\ true;\ \kappa_1^2 \oplus answer; \kappa_1^2!\langle 5\rangle; P \\
& \ \mathsf{else}\ \kappa_2^2, \kappa_3^2 \oplus\ false;\ \kappa_2^2!\langle 5\rangle; \kappa_3^2!\langle 4\rangle; \kappa_2^2?\langle y_1\rangle; \kappa_3^2?\langle y_2\rangle; X_1\langle y_1, y_2, \kappa_1^2, \kappa_2^2, \kappa_3^2\rangle \\
& \mid \kappa_2^3\&\ \{true\colon \mathbf{0}, false\colon \kappa_2^3?\langle y_1\rangle; \kappa_2^3!\langle y_1 + 1\rangle; X_2\langle\kappa_2^3\rangle\} \\
& \mid \kappa_3^4\&\ \{true\colon \mathbf{0}, false\colon \kappa_3^4?\langle y_1\rangle; \kappa_3^4!\langle y_1 - 1\rangle; X_3\langle\kappa_3^4\rangle\}) \longrightarrow^{[R-CProcCall]}_{[R-Scop],[R-IfF],[R-Label]}
\end{array}
\end{array}
$$

$$
\begin{array}{l}
\mathsf{def}\ X_1\langle y_1, y_2, x_1, x_2, x_3\rangle = Q, X_2\langle x_2\rangle = R, X_3\langle x_3\rangle = S\ \mathsf{in} \\
\begin{array}{ll}
(\nu\kappa_1, \kappa_2, \kappa_3) & (\kappa_1^1\&\{answer : \kappa_1^1?\langle y_1\rangle; 0 \\
& \mid \kappa_2^2!\langle 5\rangle; \kappa_3^2!\langle 4\rangle; \kappa_2^2?\langle y_1\rangle; \kappa_3^2?\langle y_2\rangle; X_1\langle y_1, y_2, \kappa_1^2, \kappa_2^2, \kappa_3^2\rangle \\
& \mid \kappa_2^3?\langle y\rangle; \kappa_2^3!\langle y + 1\rangle; X_2\langle\kappa_2^3\rangle \\
& \mid \kappa_3^4?\langle y\rangle; \kappa_3^4!\langle y - 1\rangle; X_3\langle\kappa_3^4\rangle) \longrightarrow^{[R-CProcCall],[R-Scop],[R-Multicast]}_{[R-Multicast]}
\end{array}
\end{array}
$$

$$
\begin{array}{l}
\mathsf{def}\ X_1\langle y_1, y_2, x_1, x_2, x_3\rangle = Q, X_2\langle x_2\rangle = R, X_3\langle x_3\rangle = S\ \mathsf{in} \\
\begin{array}{ll}
(\nu\kappa_1, \kappa_2, \kappa_3) & (\kappa_1^1\&\{answer : \kappa_1^1?\langle y_1\rangle; 0 \\
& \mid \kappa_2^2?\langle y_1\rangle; \kappa_3^2?\langle y_2\rangle; X_1\langle y_1, y_2, \kappa_1^2, \kappa_2^2, \kappa_3^2\rangle \\
& \mid \kappa_2^3!\langle 5 + 1\rangle; X_2\langle\kappa_2^3\rangle \\
& \mid \kappa_3^4!\langle 4 - 1\rangle; X_3\langle\kappa_3^4\rangle) \longrightarrow^{[R-CProcCall],[R-Scop],[R-Multicast],[R-Multicast]}
\end{array}
\end{array}
$$

$$
\mathsf{def}\ X_1\langle y_1, y_2, x_1, x_2, x_3\rangle = Q, X_2\langle x_2\rangle = R, X_3\langle x_3\rangle = S\ \mathsf{in}
$$

$$(\nu\kappa_1,\kappa_2,\kappa_3)\quad(\kappa_1^1\&\{answer : \kappa_1^1?\langle y_1\rangle; 0 \mid X_1\langle 6,3,\kappa_1^2,\kappa_2^2,\kappa_3^2\rangle \mid X_2\langle\kappa_2^3\rangle \mid X_3\langle\kappa_3^4\rangle) \longrightarrow^{[R-CProcCall],[R-Scop],[R-ProcCall],[R-ProcCall],[R-ProcCall]}$$

Reduction steps of the Delegation example Below we present the reduction steps of the Delegation example described in Section 2.4. The interesting step of this reduction is the third one where higher-order communication takes place through the same rule: [R-Multicast], that is used to send primitive values. The other reduction rules read symmetrically as in the previous example.

$$\overline{a}_{[2,3]}\,(x_1,x_2).\overline{b}_{[2]}\,(y_1).y_1!\langle x_1,x_2\rangle;\mathbf{0} \mid$$
$$a_{[2]}\,(x_1,x_2).x_2!\langle\text{“}\textit{The computer and the brain}\text{”}\rangle;\mathbf{0} \mid$$
$$a_{[3]}\,(x_1,x_2).x_1?(y');x_2?(y'');\mathbf{0} \mid$$
$$b_{[2]}\,(y_1).y_1?(y_2,y_3);y_2!\langle\text{“}\textit{John von Neumann}\text{”}\rangle;\mathbf{0} \longrightarrow^{[R-Par],[R-Link]}$$

$$(\nu\kappa_1,\kappa_2)(\overline{b}_{[2]}\,(y_1).y_1!\langle\kappa_1^1,\kappa_2^1\rangle;\mathbf{0} \mid \kappa_2^2!\langle\text{“}\textit{The computer and the brain}\text{”}\rangle;\mathbf{0} \mid \kappa_1^3?(y');\kappa_2^3?(y'');\mathbf{0}) \mid$$
$$b_{[2]}\,(y_1).y_1?(y_2,y_3);y_2!\langle\text{“}\textit{John von Neumann}\text{”}\rangle;\mathbf{0} \longrightarrow^{[R-Par],[R-Link],[R-StructC]}$$

$$(\nu\kappa_1,\kappa_2,\kappa_1',\kappa_2')(\kappa'^1!\langle\kappa_1^1,\kappa_2^1\rangle;\mathbf{0} \mid \kappa_2^2!\langle\text{“}\textit{The computer and the brain}\text{”}\rangle;\mathbf{0} \mid \kappa_1^3?(y');\kappa_2^3?(y'');\mathbf{0} \mid$$
$$\kappa'^2?(y_2,y_3);y_2!\langle\text{“}\textit{John von Neumann}\text{”}\rangle;\mathbf{0}) \longrightarrow^{[R-Par],[R-Multicast]}$$

$$(\nu\kappa_1,\kappa_2,\kappa_1',\kappa_2')(\kappa_2^2!\langle\text{“}\textit{The computer and the brain}\text{”}\rangle;\mathbf{0} \mid \kappa_1^3?(y');\kappa_2^3?(y'');\mathbf{0} \mid$$
$$\kappa_1^1!\langle\text{“}\textit{John von Neumann}\text{”}\rangle;\mathbf{0}) \longrightarrow^{[R-Par],[R-Multicast],[R-Multicast]} \mathbf{0}$$

2.6.3 Well-formedness and Linearity of Global Types

Well-formedness of global types Multicasting of values and labels may form global types that correspond to meaningless and erroneous sessions. The rules in Figure 2.8 ensure in a global type that for each receiver (of a value or label) there corresponds a channel that performs the delivery and that a behaviour is sent (as delegation) in multicast if the sub-behaviours of the decomposed behaviour are independent. Independent behaviours can be performed in any order without affecting the logic of the program as we shall see later in

$$\frac{|\tilde{\mathtt{p}}'| = |\tilde{\mathtt{n}}| \quad \emptyset \vdash \tilde{V} \quad \Gamma \vdash G}{\Gamma \vdash \mathtt{p} \to \tilde{\mathtt{p}}' : \tilde{\mathtt{n}}\,\langle \tilde{V} \rangle.G}\,\lfloor\text{WF-ExV}\rfloor \qquad \frac{|\tilde{\mathtt{p}}'| = |\tilde{\mathtt{n}}| \quad \forall i \in I.\Gamma \vdash G_i}{\Gamma \vdash \mathtt{p} \to \tilde{\mathtt{p}}' : \tilde{\mathtt{n}}\,\{l_i : G_i\}_{i \in I}}\,\lfloor\text{WF-Branch}\rfloor$$

$$\frac{|\tilde{\mathtt{p}}'| = |\tilde{\mathtt{n}}| = |\tilde{T}| \quad T_1 \neq T_2 \neq \cdots T_{\mathrm{m}} \quad \Gamma \vdash G}{\Gamma \vdash \mathtt{p} \to \tilde{\mathtt{p}}' : \tilde{\mathtt{n}}\,\langle \tilde{T}@\mathtt{p}'' \rangle.G}\,\tilde{T} = T_1, ..., T_{\mathrm{m}}\lfloor\text{WF-ExC}\rfloor$$

$$\frac{\Gamma, \mathbf{t} \vdash G}{\Gamma \vdash \mu\mathbf{t}.G}\,\lfloor\text{WF-Rec}\rfloor \qquad \frac{}{\Gamma, \mathbf{t} \vdash \mathbf{t}}\,\lfloor\text{WF-Var}\rfloor \qquad \frac{}{\Gamma \vdash \mathsf{end}}\,\lfloor\text{WF-End}\rfloor$$

$$\frac{}{\Gamma \vdash \mathsf{bool}}\,\lfloor\text{WF-Bool}\rfloor \qquad \frac{}{\Gamma \vdash \mathsf{nat}}\,\lfloor\text{WF-Nat}\rfloor \qquad \frac{\Gamma \vdash G}{\Gamma \vdash \langle G \rangle}\,\lfloor\text{WF-Name}\rfloor$$

Figure 2.8: CSMS: Well-formedness of global types.

Section 2.6.4. For example, the two end-point types $n!\langle U \rangle$ and $n'!\langle U \rangle$ are independent as the order of actions they abstract does not affect the logic of the program. In contrast, the order of $n!\langle U \rangle$ and $n'?\langle U \rangle$ breaks the order of causalities, leading to erroneous behaviour, as in the global type:

$$\mathtt{p} \to \mathtt{p}' : \mathrm{n}\,\langle U \rangle.\mathtt{p}' \to \mathtt{p} : \mathrm{n}'\,\langle U \rangle$$

where the message sent in the second causality is a manipulation of the message sent in the first causality. The utility in practice of multicast delegation is debatable: the scenarios expressed are not of interest in practice, but it represents an interesting mathematical problem which the author thought worth investigating. Variable context for well-formedness is defined as a list of type variables, $\Gamma ::= \Gamma, \mathbf{t}$. The rules are of the form $\Gamma \vdash G$, read, "In the variable typing Γ, global type G is well-formed".

Rule $\lfloor$WF-ExV$\rfloor$ ensures that the cardinality of the receiver set is equal to the cardinality of the channel set, the message type is well-formed and that the inductive part is well-formed. The context of well-formedness for the message type judgement is empty to restrict the scope of infinite behaviour to the global type checked and not to other global types carried in the message exchange; e.g. the global type below:

$$\mu\mathbf{t}.\mathtt{p} \to \mathtt{p}' : \mathrm{n}\,\langle\langle \mathbf{t} \rangle\rangle.\mathsf{end}$$

defines the type variable $\mathbf{t}$ as the global type of a shared name exchanged between the two participants $\mathtt{p}$ and $\mathtt{p}'$. In this study, we follow the equi-recursive approach to the relation between the recursive type $\mu\mathbf{t}.G$ and its one-time unfolding; i.e., the former and the latter

are equal. Therefore, the above global type is equal to

$$\mathtt{p} \to \mathtt{p}' \colon \mathrm{n}\,\langle\langle \mu \mathbf{t}.\mathtt{p} \to \mathtt{p}' \colon \mathrm{n}\,\langle\langle \mathbf{t} \rangle\rangle.\mathsf{end}\rangle\rangle.\mathsf{end}.$$

This global type captures the exchange of the shared name of the session as described below:

$$\bar{a}[2](x).x!\langle a\rangle;\mathbf{0} \mid a[2](x).x?\langle y\rangle;\mathbf{0}$$

However this example is not well-typed by the above global type according to our type system as we shall see later in Section 2.6.4. Thus, we restrict the set of global types that express correct sessions to maintain a typing coherence between well-formed global types and well-typed programs. This problem was not addressed in the initial work of multiparty session types by Honda et al. (2008b).

Rule ⌊WF-Branch⌋ checks that the cardinality of the receiver set is equal to the cardinality of the channel set and that the inductive parts are well-formed.

Rule ⌊WF-ExC⌋ ensures that the cardinality of the receiver set, channel set and end-point types set are equal, the end-point types are independent and that the inductive part is well-formed. The full definition of independent end-point types is given in Section 2.6.4 where end-points types are defined.

Other rules check that the inductive global type is well-formed, look up for type variable in the context and define end as a well-formed global type.

Linearity of channels Recollecting from Section 2.5: programming multiparty sessions without race conditions on channels requires the definition of all the communications dependencies between all the participants. Global types *per se* do not guarantee programs from not having race conditions and so, a linearity property is defined to check for race conditions the channels of a global type and so, later type-check the program.

Below we define an order relation of communications in a global type. The ordering relation will be used later to define the linearity property. Before that we define the notion of prefix.

Definition 2.6.6 (prefix)**.** We say the initials "$\mathtt{p} \to \mathtt{p}_i : \mathrm{m}_i$" in $\mathtt{p} \to \tilde{\mathtt{p}}' \colon \tilde{\mathrm{m}}\,\langle \tilde{U} \rangle.G'$ and $\mathtt{p} \to \tilde{\mathtt{p}}' \colon \tilde{\mathrm{m}}\,\{l_i : G_i\}_{i \in I}$ for all $\mathtt{p}_i \in \tilde{\mathtt{p}}', \mathrm{m}_i \in \tilde{\mathrm{m}}$ are called *prefixes from* $\mathtt{p}$ *to* $\mathtt{p}_\mathtt{i}$ *at* m_i *over* G' and $\{G_i\}_{i \in I}$, where in the former $\tilde{U}$ is called a *carried type*. If $\tilde{U}$ is a carried type in a prefix in G then $\tilde{U}$ is also a carried type in G.

Conventions 2.6.7. We assume that in each prefix from $\mathtt{p}$ to $\mathtt{p}'$ we have $\mathtt{p} \neq \mathtt{p}'$, i.e. we prohibit reflexive interaction.

Definition 2.6.8 (prefix ordering)**.** Write $\phi, \phi', ..$ for prefixes occurring in a global type, say G (but not in its carried types), seen as nodes of G as a graph. We write $\phi \in G$ when ϕ occurs in G. Then we write $\phi_1 \prec \phi_2 \in G$ when ϕ_1 directly or indirectly prefixes ϕ_2 in G. Formally $\prec$ is the least partial order generated by:

$$\begin{array}{c} \phi_i \prec \phi \in \mathtt{p} \to \tilde{\mathtt{p}}' \colon \tilde{\mathtt{m}} \langle \tilde{U} \rangle . G' \quad \text{if } \phi_i = \mathtt{p} \to \mathtt{p}_i : \mathrm{m}_i, \phi \in G' \text{ and } \mathtt{p}_i \in \tilde{\mathtt{p}}', \mathrm{m}_i \in \tilde{\mathtt{m}} \\ \phi_i \prec \phi \in \mathtt{p} \to \tilde{\mathtt{p}}' \colon \tilde{\mathtt{m}} \{ l_i : G_i \}_{i \in I} \quad \text{if } \phi_i = \mathtt{p} \to \mathtt{p}_i : \mathrm{m}_i \text{ and } \exists h \in I . \phi \in G_h \, \mathtt{p}_i \in \tilde{\mathtt{p}}', \mathrm{m}_i \in \tilde{\mathtt{m}} \end{array}$$

Further we set $\phi_1 \prec \phi_2 \in G$ if $\phi_1 \prec \phi_2 \in G'$ and G' occurs in G but not in its carried types.

Consider a global type:

$$\mathtt{A} \to \mathtt{B} \colon \mathrm{m}_1 \langle U \rangle \mathtt{A} \to \mathtt{C} \colon \mathrm{m}_2 \langle U' \rangle \mathsf{end}$$

The two prefixes are ordered by $\prec, \mathtt{A} \to \mathtt{B} : \mathrm{m}_1 \prec \mathtt{A} \to \mathtt{C} : \mathrm{m}_2$. This ordering means "only after the first sending and receiving take place, the second sending and receiving take place". This interpretation is valid for calculi of synchronous communications and not for asynchronous ones; e.g. in the asynchronous calculus of Honda et al. (2008b), $\mathtt{C}$ may receive its message before $\mathtt{B}$.

Below we provide the definition of dependency between two consecutive causalities and the definition of input and output dependency, following the causality analysis given in Section 2.5. The definitions are a symbolic representation of the diagrams introduced in the same section.

Definition 2.6.9. (dependency relations) Fix G. The relation $\prec_\iota$, with $\iota \in \{\mathsf{II}, \mathsf{IO}, \mathsf{OI}, \mathsf{OO}\}$, over its prefixes is generated from:

$$\begin{array}{ll} \phi_1 \prec_{\mathtt{II}} \phi_2 & \text{if } \phi_1 \prec \phi_2 \text{ and } \phi_i = \mathtt{p}_i \to \mathtt{p} : \mathrm{m}_i \ (i = 1, 2) \\ \phi_1 \prec_{\mathtt{IO}} \phi_2 & \text{if } \phi_1 \prec \phi_2, \phi_1 = \mathtt{p}_1 \to \mathtt{p} : \mathrm{m}_1 \text{ and } \phi_2 = \mathtt{p} \to \mathtt{p}_2 : \mathrm{m}_2. \\ \phi_1 \prec_{\mathtt{OI}} \phi_2 & \text{if } \phi_1 \prec \phi_2, \phi_1 = \mathtt{p} \to \mathtt{p}_1 : \mathrm{m}_1 \text{ and } \phi_2 = \mathtt{p}_2 \to \mathtt{p} : \mathrm{m}_2. \\ \phi_1 \prec_{\mathtt{OO}} \phi_2 & \text{if } \phi_1 \prec \phi_2, \phi_i = \mathtt{p} \to \mathtt{p}_i : \mathrm{m}_i \ (i = 1, 2) \end{array}$$

An *input dependency* from ϕ_1 to ϕ_2 is a chain of the form $\phi_1 \prec_{\iota_1} \cdots \prec_{\iota_n} \phi_2$ $(n \geq 2)$ such that if

$$\iota_i \in \{\mathsf{OI}, \mathsf{II}\} \text{ then } \iota_{i+1} \in \{\mathsf{OO}, \mathsf{OI}\} \textit{ or}$$
$$\iota_i \in \{\mathsf{IO}, \mathsf{OO}\} \text{ then } \iota_{i+1} \in \{\mathsf{II}, \mathsf{IO}\}$$

for $1 \leq i \leq n-2$ and $\iota_n \in \{\mathsf{II}, \mathsf{OI}\}$.

An *output dependency* from ϕ_1 to ϕ_2 is a chain of the form $\phi_1 \prec_{\iota_1} \cdots \prec_{\iota_n} \phi_2$ $(n \geq 2)$ such that if

$$\iota_i \in \{\mathsf{OI}, \mathsf{II}\} \text{ then } \iota_{i+1} \in \{\mathsf{OO}, \mathsf{OI}\} \textit{ or}$$
$$\iota_i \in \{\mathsf{IO}, \mathsf{OO}\} \text{ then } \iota_{i+1} \in \{\mathsf{II}, \mathsf{IO}\}$$

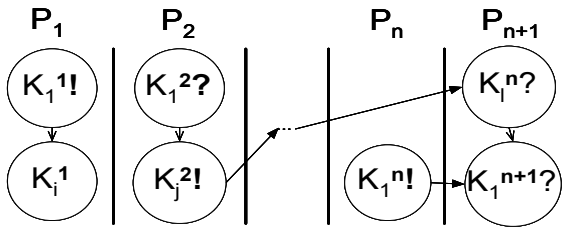
for $1 \leq i \leq n-2$ and $\iota_n \in \{\mathsf{OO}, \mathsf{IO}\}$.

For the sake of the argument regarding the formal definition of linearity in the initial work of multiparty session types, discussed in Section 2.5, we provide the formal definition of the *input dependency* as in that work and our version adapting the intuition given in Section 2.5 to asynchronous communications:

- *An input dependency from ϕ_1 to ϕ_2 is a chain of the form $\phi_1 \prec_{\iota_1} \cdots \prec_{\iota_n} \phi_2$ $(n \geq 0)$ such that $\phi_i \in \{\mathsf{II}, \mathsf{IO}\}$ for $1 \leq i \leq n-1$ and $\phi_n = \mathsf{II}$.* (as in Honda et al. (2008b))

- *An input dependency from ϕ_1 to ϕ_2 is a chain of the form $\phi_1 \prec_{\iota_1} \cdots \prec_{\iota_n} \phi_2$ $(n \geq 0)$ such that $\phi_i \in \{\mathsf{IO}\}$ for $1 \leq i \leq n-1$ and $\phi_n = \mathsf{II}$.* (our version)

The condition $\phi_i \in \{\mathsf{II}\}$ is unnecessary to define a relation between the first communication and the action that precedes the receiving action of the second communication, only the $\phi_i \in \{\mathsf{IO}\}$ is sufficient, resulting in an longer input chain than our version. The same result is valid for the output chain. Below, we provide also the diagram for the input dependency of asynchronous communications:



Definition 2.6.10. (linearity) Let G be well-formed. G is *linear* if, whenever $\phi_i = \mathtt{p}_i \to \mathtt{p}_i' : \mathrm{m}$ $(i = 1, 2)$ are in G for some m and do not occur in different branches of a branching, then both input and output dependencies exist from ϕ_1 to ϕ_2. In case of multicasting (values or labels), all the chains achieved by distributing each prefix of multicasting on the rest of G have to be checked if they satisfy the above conditions. If G carries other global types, we inductively demand the same.

We illustrate the condition on branching with an example:

$$\begin{array}{ll} 1. & \mathtt{A} \to \mathtt{B} : \mathrm{m}\{ok : \mathtt{C} \to \mathtt{D} : \mathrm{m}_1.\mathsf{end} \\ 2. & \qquad\qquad quit : \mathtt{C} \to \mathtt{D} : \mathrm{m}_1.\mathsf{end}\ \} \end{array}$$

The type represents branching: since only one of two branches is selected, there is no conflict between the two prefixes $\mathtt{C} \to \mathtt{D} : \mathrm{m}_1$ in Lines 1 and 2.

Linearity and its violation can be detected algorithmically, without infinite unfoldings. First we observe we do need to unfold once. The global type below is linear in its 0-th unfolding (i.e. we replace $\mathbf{t}$ with end):

$$\mu\mathbf{t}.\mathtt{A} \to \mathtt{B} : \mathrm{m}.\mathtt{B} \to \mathtt{C} : \mathrm{n}.\mathtt{A} \to \mathtt{C} : \mathrm{m}.\mathbf{t}$$

but when unfolded once, it becomes non-linear, as witnessed by:

$$\mathtt{A} \to \mathtt{B} : \mathrm{m}.\mathtt{B} \to \mathtt{C} : \mathrm{n}.\mathtt{A} \to \mathtt{C} : \mathrm{m}.\mu\mathbf{t}.\mathtt{A} \to \mathtt{B} : \mathrm{m}.\mathtt{B} \to \mathtt{C} : \mathrm{n}.\mathtt{A} \to \mathtt{C} : \mathrm{m}.\mathbf{t}$$

since there are no input and output dependencies between $\mathtt{A} \to \mathtt{C} : \mathrm{m}$ and $\mathtt{A} \to \mathtt{B} : \mathrm{m}$. But in fact unfolding once turns out to be enough. Given G, let us call the *one-time unfolding of* G the result of unfolding once for each recursion in G (but never in carried types), and replacing the remaining variable with end.

Notation 2.6.11. In the following, we write $G(0)$, $G(1)$, ..., $G(n)$, ... for the result of n-times unfolding of each recursion in G. For example if G is $\mu\mathbf{t}.G'$ and this is the only recursion in G, then $G(0)$ is given as $G'[\mathsf{end}/\mathbf{t}]$, $G(1)$ is given as $G'[G(0)/\mathbf{t}]$ and, for each n, $G(n+1)$ is given as $G'[G(n)/\mathbf{t}]$. If G contains more than one recursion we perform the unfolding of each of its recursions.

Proposition 2.6.12. (1) The one-time unfolding of a global type is linear iff its n-th unfolding is linear. (2) The linearity of a global type is decidable at worst case in cubic time-complexity.

$T ::=$		End-point Types:	$\mid$	$\mathrm{m}\&\{l_i : T_i\}_{i \in I}$	Branching
	$\tilde{\mathrm{m}}!\langle \tilde{U} \rangle; T$	Output	$\mid$	$\mu \mathbf{t}.\mathrm{T}$	Recursion
$\mid$	$\mathrm{m}?\langle \tilde{U} \rangle; T$	Input	$\mid$	$\mathbf{t}$	Type variable
$\mid$	$\tilde{\mathrm{m}} \oplus \langle l \rangle; T$	Selection	$\mid$	end	End

Figure 2.9: CSMS: Syntax of end-point types.

Proof. The proof of (1) is given in Honda et al. (2008a), where each n-time unfolding is linear if one-time unfolding is linear by induction on n. The other side of the proposition: if the n-th unfolding of a global time is linear, then its one-time unfolding is linear is true by definition of n-time unfolding. Since the proof is not a direct contribution of the author of this thesis, we omit it from the latter.

(2) is an immediate corollary of (1). See Appendix A.1 for the linearity algorithm and its computational complexity analysis. □

2.6.4 Type System

End-point Session Types End-point session types (see Figure 2.9) or end-point types, as we refer to them throughout this thesis, capture the behaviour of a process. Session type $\tilde{\mathrm{m}}!\langle \tilde{U} \rangle; T$ represents all processes that send single or multiple value of type $\tilde{U}$ on channels indexed by $\tilde{\mathrm{m}}$ and that the rest of behaviour is represented by T. Session type $\mathrm{m}?\langle \tilde{U} \rangle; T$ represents all processes that receive single or multiple value of type $\tilde{U}$ on channel indexed by m and that the rest of behaviour is represented by T. Session type $\tilde{\mathrm{m}} \oplus \langle l \rangle; T$ represents all processes that send a label l and that the rest of behaviour is represented by T. Session type $\mathrm{m}\&\{l_i : T_i\}_{i \in I}$ represents all processes that receive one of the l_i labels and that the rest of behaviour is represented by T_i. Session type $\mu \mathbf{t}.\mathrm{T}$ represents all processes that have a recursive behaviour captured by T. Session type end represents the $\mathbf{0}$ process. The type U represents the same set of values as in global types.

Definition 2.6.13. Two end-point types T_1 and T_2 are independent ($\neq$), if and only if each action type of one is independent from every action type of the other.

Definition 2.6.14. Independent relation ($\neq$) between two action types is defined as:

$$\mathrm{n}!\langle U \rangle \neq \mathrm{m}!\langle U' \rangle$$
$$\mathrm{n}?\langle U \rangle \neq \mathrm{m}?\langle U' \rangle$$

Projection and Coherence This paragraph defines the projection of a global type over its participants. The result is the end-point types that type-check the processes of the session.

Definition 2.6.15 (Projection)**.** Let G be linear. The *projection of G onto* $\mathtt{q}$, written $G \upharpoonright \mathtt{q}$, is inductively given as:

$$\mathtt{p} \to \tilde{\mathtt{p}}' \colon \tilde{\mathrm{m}}\, \langle \tilde{V} \rangle . G' \upharpoonright \mathtt{q} \;=\; \begin{cases} \tilde{\mathrm{m}}! \langle \tilde{V} \rangle ; (G' \upharpoonright \mathtt{q}) & \textit{if } \mathtt{q} = \mathtt{p} \textit{ and } \mathtt{q} \notin \tilde{\mathtt{p}}' \\ \mathrm{m}_i ? \langle \tilde{V} \rangle ; (G' \upharpoonright \mathtt{q}) & \textit{if } \mathtt{q} \in \tilde{\mathtt{p}}' (\text{such that } \mathtt{p}'_i = \mathtt{q}) \textit{ and } \mathrm{m}_i \in \tilde{\mathrm{m}} \textit{ and } \mathtt{q} \neq \mathtt{p} \\ G' \upharpoonright \mathtt{q} & \textit{if } \mathtt{q} \notin \tilde{\mathtt{p}}' \textit{ and } \mathtt{q} \neq \mathtt{p} \end{cases}$$

$$\mathtt{p} \to \tilde{\mathtt{p}}' \colon \tilde{\mathrm{m}}\, \langle \tilde{T} @ \mathtt{p}'' \rangle . G' \upharpoonright \mathtt{q} \;=\; \begin{cases} \tilde{\mathrm{m}}! \langle \tilde{T} @ \mathtt{p}'' \rangle ; (G' \upharpoonright \mathtt{q}) & \textit{if } \mathtt{q} = \mathtt{p} \textit{ and } \mathtt{q} \notin \tilde{\mathtt{p}}' \\ \mathrm{m}_i ? \langle T_i @ \mathtt{p}'' \rangle ; (G' \upharpoonright \mathtt{q}) & \textit{if } \mathtt{q} \in \tilde{\mathtt{p}}' (\text{such that } \mathtt{p}'_i = \mathtt{q}) \textit{ and } \mathrm{m}_i \in \mathrm{m} \textit{ and} \\ & \quad T_i \in \widetilde{T} \textit{ and } \mathtt{q} \neq \mathtt{p} \\ (G' \upharpoonright \mathtt{q}) & \textit{if } \mathtt{q} \notin \tilde{\mathtt{p}}' \textit{ and } \mathtt{q} \neq \mathtt{p} \end{cases}$$

$$\mathtt{p} \to \tilde{\mathtt{p}}' \colon \tilde{\mathrm{m}}\, \{ l_i : G_i \}_{i \in I} \upharpoonright \mathtt{q} \;=\; \begin{cases} \tilde{\mathrm{m}} \oplus \{ l_i : G_i \upharpoonright \mathtt{q} \}_{i \in I} & \textit{if } \mathtt{q} = \mathtt{p} \textit{ and } \mathtt{q} \notin \tilde{\mathtt{p}}' \\ \mathrm{m}_j \& \{ l_i : G_i \upharpoonright \mathtt{q} \}_{i \in I} & \textit{if } \mathtt{q} \in \tilde{\mathtt{p}}' (\text{such that } \mathtt{p}'_j = \mathtt{q} \text{ and } \mathrm{m}_j \in \tilde{\mathrm{m}}) \textit{ and } \mathtt{q} \neq \mathtt{p} \\ \sqcup_{i \in I} G_i \upharpoonright \mathtt{q} & \textit{if } \mathtt{q} \notin \tilde{\mathtt{p}}' \textit{ and } \mathtt{q} \neq \mathtt{p} \textit{ and } \forall i, j \in I . G_i \upharpoonright \mathtt{q} \bowtie G_j \upharpoonright \mathtt{q} \end{cases}$$

$$\mu \mathbf{t} . G \upharpoonright \mathtt{q} \;=\; \mu \mathbf{t} . (G \upharpoonright \mathtt{q}) \quad \mathbf{t} \upharpoonright \mathtt{q} = \mathbf{t}, \quad \mathsf{end} \upharpoonright \mathtt{q} = \mathsf{end}$$

When a side condition does not hold, the map is undefined.

The mapping is intuitive. In the receiving case of delegation, the type received is the i-th element of the list $\widetilde{T}@\mathtt{p}''$, corresponding to the same index as the channel. This is to support multicast delegation of a behaviour, decomposed into independent components and each of them sent to a member of the multicast group. In branching, in the case when $\mathtt{q}$ is not equal either to $\mathtt{p}$ or to $\mathtt{p}'$, all inductive projections of $\mathtt{q}$ should return an identical role type up to mergeability $\bowtie$. The notion of mergeability is introduced by Carbone et al. (2009) as an equivalence relation over end-point types. Intuitively, two different &

end-point types are mergeable if denoted by different labels; e.g. the projection of global type:

$$\mathtt{p} \to \mathtt{p}' : \mathrm{m} \left\{ \begin{array}{l} \mathit{true} : \mathtt{p}' \to \mathtt{q} : \mathrm{n}\{\mathit{true} : G\}, \\ \mathit{false} : \mathtt{p}' \to \mathtt{q} : \mathrm{n}\{\mathit{false} : G'\} \end{array} \right\}$$

onto $\mathtt{q}$ returns $\&\mathrm{n}\{\mathit{true}{:}G \upharpoonright \mathtt{q}, \mathit{false}{:}G \upharpoonright \mathtt{q}\}$, where $G \upharpoonright \mathtt{q} \neq G' \upharpoonright \mathtt{q}$. Below, we give the formal definition as given by Carbone et al. (2009).

Definition 2.6.16 (Mergebility)**.** *If* $\forall i \in (I \cap J).T_i \bowtie T'_i$ *and* $\forall i \in I \setminus J.\forall j \in J \setminus I.l_i \neq l_j$ *then* $\&\langle \mathtt{p}, \{l_i : T_i\}_{i\in I}\rangle \bowtie \&\langle \mathtt{p}, \{l_j : T'_j\}_{j\in J}\rangle$.
$\&\langle \mathtt{p}, \{l_i : T_i\}\rangle \sqcup \&\langle \mathtt{p}, \{l_j : T'_j\}_{j\in J}\rangle = \&\langle \mathtt{p}, \{l_i : T_i \sqcup T'_i\}_{i\in I\cap J} \cup \{l_i : T_i\}_{i\in I\setminus J} \cup \{l_i : T'_i\}_{i\in J\setminus I}\rangle$.

Below $\mathsf{pid}(G)$ denotes the set of participant numbers occurring in G (but not in carried types).

Definition 2.6.17 (Coherence)**.** (1) We say G is *coherent* if it is linear and $G \upharpoonright \mathtt{p}$ is well-defined for each $\mathtt{p} \in \mathsf{pid}(G)$, similarly for each carried global type inductively. (2) $\{T_{\mathtt{p}}@\mathtt{p}\}_{\mathtt{p}\in I}$ is *coherent* if for some coherent G s.t. $I = \mathsf{pid}(G)$, we have $G \upharpoonright \mathtt{p} = T_{\mathtt{p}}$ for each $\mathtt{p} \in I$.

Theorem 2.6.18. Coherence of G is decidable at the worst case in cubic $O(n^3)$ time complexity.

Proof. By Theorem 2.6.12 (2). See Appendix A.2 for algorithm and computational complexity analysis. □

Typing Relation The typing relation of processes, written $\Gamma \vdash P \triangleright \Delta$, read "in the environment Γ, process P has type Δ", is defined by a set of axiom and inference rules assigning types to terms, given in Figure 2.10. The typing context Γ is a sequence of variables, shared names and their types V (see Figure 2.4) and the type Δ is a sequence of channels and their types $T@\mathtt{p}$ (see Figure 2.9). We write $\mathit{dom}(\Gamma)$ and $\mathit{dom}(\Delta)$ for the set of respectively values and channels bound in Γ and Δ. In the case when the type has the shape $\langle G\rangle$ then we assume that G is coherent. The context is extended in the right side through the "comma" operator with elements not occurring in the context. We define the grammar of typing context and typings as follows. Below in "$\Gamma, u : V$", we assume u does not occur in Γ and in "$\Delta, \tilde{k} : \widetilde{T}@\mathtt{p}$", we assume *no* channel in $\tilde{k}$ occurs in the domain of Δ.

$$\Gamma, u\colon\langle G\rangle \vdash u\colon\langle G\rangle \qquad \Gamma \vdash \mathbf{true}, \mathbf{false}\colon \mathsf{bool} \qquad \frac{\Gamma \vdash e_i \triangleright \mathsf{bool}}{\Gamma \vdash e_1 \mathtt{or}\ e_2\colon \mathsf{bool}}$$

[Name], [Bool], [Or]

$$\frac{\Gamma \vdash u\colon\langle G\rangle \quad \Gamma \vdash P \triangleright \Delta, \tilde{x}\colon (G\upharpoonright 1)@1 \quad |\tilde{x}| = |\mathsf{sid}(G)|}{\Gamma \vdash \bar{u}[2..\mathrm{n}](\tilde{x}).P \triangleright \Delta} \quad [\textsc{Mcast}]$$

$$\frac{\Gamma \vdash u\colon\langle G\rangle \quad \Gamma \vdash P \triangleright \Delta, \tilde{x}\colon (G\upharpoonright p)@\mathtt{p} \quad |\tilde{x}| = |\mathsf{sid}(G)|}{\Gamma \vdash u[\mathtt{p}](\tilde{x}).P \triangleright \Delta} \quad [\textsc{Macc}]$$

$$\frac{\Gamma \vdash \tilde{e}\colon \tilde{V} \qquad \Gamma \vdash P \triangleright \Delta, \tilde{k}\colon T@\mathtt{p}}{\Gamma \vdash k_{\tilde{\mathrm{m}}}!\langle\tilde{e}\rangle; P \triangleright \Delta, \tilde{k}\colon \tilde{\mathrm{m}}!\langle\tilde{V}\rangle; T@\mathtt{p}} \quad [\textsc{Send}]$$

$$\frac{\Gamma, \tilde{y}\colon \tilde{V} \vdash P \triangleright \Delta, \tilde{k}\colon T@\mathtt{p}}{\Gamma \vdash k_{\mathrm{m}}?\langle\tilde{y}\rangle; P \triangleright \Delta, \tilde{k}\colon \mathrm{m}?\langle\tilde{V}\rangle; T@\mathtt{p}} \quad [\textsc{Rcv}]$$

$$\frac{\Gamma \vdash P \triangleright \Delta, \tilde{k}\colon T@\mathtt{p}}{\Gamma \vdash k_{\tilde{\mathrm{m}}}!\langle\tilde{t}\rangle; P \triangleright \Delta, \tilde{k}\colon \tilde{\mathrm{m}}!\langle\widetilde{T'}@p'\rangle; T@\mathtt{p},\ \tilde{t}\colon seq(\widetilde{T'})@\mathtt{p}'} \quad [\textsc{Thr}]$$

$$\frac{\Gamma \vdash P \triangleright \Delta, \tilde{k}\colon T@\mathtt{p},\ \tilde{y}\colon T'@\mathtt{p}'}{\Gamma \vdash k_{\mathrm{m}}?\langle\tilde{y}\rangle; P \triangleright \Delta, \tilde{k}\colon \mathrm{m}?\langle T'@\mathtt{p}'\rangle; T@\mathtt{p}} \quad [\textsc{Cat}]$$

$$\frac{\Gamma \vdash P \triangleright \Delta, \tilde{k}\colon T_j@\mathtt{p} \qquad j \in I}{\Gamma \vdash k_{\tilde{\mathrm{m}}} \oplus \langle l_j\rangle; P \triangleright \Delta, \tilde{k} : \tilde{\mathrm{m}} \oplus \{l_i : T_i\}_{i\in I}@\mathtt{p}} \quad [\textsc{Sel}]$$

$$\frac{\Gamma \vdash P_i \triangleright \Delta, \tilde{k}\colon T_i@\mathtt{p} \qquad \forall i \in I}{\Gamma \vdash\ k_{\mathrm{m}}\&\{l_i : P_i\}_{i\in I} \triangleright \Delta, \tilde{k}\colon \mathrm{m}\ \&\{l_i : T_i\}_{i\in I}@\mathtt{p}} \quad [\textsc{Br}]$$

$$\frac{\Gamma \vdash P \triangleright \Delta \qquad \Gamma \vdash Q \triangleright \Delta' \qquad \Delta \asymp \Delta'}{\Gamma \vdash P \mid Q \triangleright \Delta \circ \Delta'} \quad [\textsc{Conc}]$$

$$\frac{\Gamma \vdash e \triangleright \mathsf{bool} \qquad \Gamma \vdash P \triangleright \Delta \qquad \Gamma \vdash Q \triangleright \Delta}{\Gamma \vdash \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \triangleright \Delta} \quad [\textsc{If}]$$

$$\frac{\Gamma \vdash P \triangleright \Delta \qquad \Delta <: \Delta'}{\Gamma \vdash P \triangleright \Delta'} \qquad \frac{\Delta\ \mathsf{end\ only}}{\Gamma \vdash \mathbf{0} \triangleright \Delta} \quad [<:], [\textsc{Inact}]$$

$$\frac{\Gamma, a\colon\langle G\rangle \vdash P \triangleright \Delta}{\Gamma \vdash (\nu a)P \triangleright \Delta} \qquad \frac{\Gamma \vdash P \triangleright \Delta, \tilde{\kappa}^{\mathtt{p}_1} : T_1@\mathtt{p}_1 \circ \ldots \circ \tilde{\kappa}^{\mathtt{p}_n} : T_n@\mathtt{p}_n}{\Gamma \vdash (\nu\tilde{\kappa})P \triangleright \Delta} \quad [\textsc{NRes}], [\textsc{CRes}]$$

$$\frac{\Gamma \vdash \tilde{e}\colon \tilde{V} \qquad \Delta\ \mathsf{end\ only}}{\Gamma, X\colon \tilde{V}\tilde{T} \vdash X\langle\tilde{e}, \tilde{k}\rangle \triangleright \Delta, \tilde{k} : \tilde{T}@\tilde{\mathtt{p}}} \quad [\textsc{Var}]$$

$$\frac{\begin{array}{c}\Gamma, X\colon \tilde{V}\tilde{T}, \tilde{y} : \tilde{V} \vdash P \triangleright \tilde{y'} : \tilde{T}@\tilde{\mathtt{p}} \\ \Gamma, X\colon \tilde{V}\tilde{T} \vdash Q \triangleright \Delta\end{array}}{\Gamma \vdash \mathsf{def}\ X(\tilde{y}, \tilde{y'}) = P\ \mathsf{in}\ Q \triangleright \Delta} \quad [\textsc{Def}]$$

Figure 2.10: CSMS: Typing relation for expressions and processes.

$$\begin{array}{rcl} \Gamma & ::= & \emptyset \mid \Gamma, a : V \mid \Gamma, X : \tilde{V}\tilde{T} \\ \Delta & ::= & \emptyset \mid \Delta, \tilde{k} : \widetilde{T}@\mathtt{p} \end{array}$$

Rule [Name] assigns a global type to a session identifier. Rules [Bool] and [Or] assign the type bool to the Boolean constants **true** and **false** and to a "or" expression based on the types of its subexpressions that both must evaluate to a Boolean.

Rules [MCast] and [MAcc] assign a type to a process prefixed by a session initiation construct based on the typing of the process: the session identifier must be typed by a global type G, the process P must be typed by the projection of the global type onto the participant in the prefix (1 in case of [MCast]) and the cardinality of the set of channels of P and G is the same. $\mathsf{sid}(G)$ denotes the set of channel in G. The type assigned to the prefixed process, Δ without the typing of the process, means that the process does not evaluate to any interactions on session u at this point. The metavariable Δ in the typing of P and the prefixed process denotes the typing of other possible sessions running in them.

Rules [Send] and [Rcv] assign a type to a process respectively prefixed by a sending and receiving action based on the typing of the subprocess: for the sending prefix, the expressions sent $\tilde{e}$ must evaluate to values of type $\tilde{V}$ and P must be well-typed by an end-point type T, while for the receiving prefix P must be well-typed by a type T over an extended context with the place-holders of values. In rule [Send] $k_{\tilde{\mathrm{m}}}$ denotes a sequence of multiple channels $k_{\mathrm{m}_1}, ..., k_{\mathrm{m}_n}$ to represent multicast send of values. In both rules, the type of P is extended in the left side with the type of the prefixed action, expressing not only the type of the message V (sent or received) but also the channels of the communication $\tilde{\mathrm{m}}$ or m.

Rules [Thr] and [Cat] assign a type to a process respectively prefixed by a sending and receiving channels action based on the typing of the subprocess similarly as in the rules [Send] and [Rcv]. In contrast to [Send], rule [Thr] defines the typing of channels not in the premises but in the conclusion type. In the [Cat] rule, the typing of the channels are present in the typing of the premiss but removed in the typing of conclusion to maintain compatible typings when the two processes are composed in parallel. $seq(T_1, ..., T_n)$ denotes the sequential composition of end-point types $T_1, .., T_n$ into $T_1; ...; T_n$.

Rules [Sel] and [Br] assign a type to respectively a process prefixed by selection and branching process based on the typing of the subprocesses: for selection, P must be well-typed by an end-point T_j such that T_j is prefixed by the label selected, while for

branching every process labelled must be well-typed by an end-point type T_i. The resulting type is the set of label and end-point type prefixed by the channels performing the action and symbols $\oplus, \&$ denoting respectively the "selection" and "branching" operation.

Rule [Conc] assigns a type to two processes composed in parallel based on their respective typing: each subprocess must be well-typed and the two types must be disjoint (compatible).

Definition 2.6.19. A partial operator $\circ$ is defined as:

$$\{T_{\mathtt{p}}@\mathtt{p}\}_{\mathtt{p}\in I} \circ \{T'_{\mathtt{p}'}@\mathtt{p}'\}_{\mathtt{p}'\in J} = \{T_{\mathtt{p}}@\mathtt{p}\}_{\mathtt{p}\in I} \cup \{T'_{\mathtt{p}'}@\mathtt{p}'\}_{\mathtt{p}'\in J}$$

if $I \cap J = \emptyset$. Then we say Δ_1 and Δ_2 are *compatible*, written $\Delta_1 \asymp \Delta_2$, if for all $\tilde{\kappa}^{\mathtt{p}} \in dom(\Delta_1)$ and $\tilde{\kappa}^{\mathtt{q}} \in dom(\Delta_2)$ such that $\tilde{\kappa} = \tilde{\kappa}^{\mathtt{p}} = \tilde{\kappa}^{\mathtt{q}}$ and $\Delta_1(\tilde{\kappa}^{\mathtt{p}}) \circ \Delta_2(\tilde{\kappa}^{\mathtt{q}})$ is defined. When $\Delta_1 \asymp \Delta_2$, the *composition of* Δ_1 *and* Δ_2, written $\Delta_1 \circ \Delta_2$, is given as:

$$\begin{array}{rcl} \Delta_1 \circ \Delta_2 & = & \{\tilde{\kappa}_{\mathtt{p}}, \tilde{\kappa}_{\mathtt{q}} : \Delta_1(\tilde{\kappa}^{\mathtt{p}}) \circ \Delta_2(\tilde{\kappa}^{\mathtt{q}}) \mid \tilde{\kappa} \in dom(\Delta_1) \cap dom(\Delta_2)\} \\ & & \{\tilde{t} : \Delta_i(\tilde{t}); \Delta_j(\tilde{t}) \mid \tilde{t} \in dom(\Delta_1) \cap dom(\Delta_2), i, j \in \{1,2\}, \Delta_i(\tilde{t}) \neq \Delta_j(\tilde{t})\} \\ & & \cup \Delta_1 \setminus dom(\Delta_2) \cup \Delta_2 \setminus dom(\Delta_1) \end{array}$$

Rules [If], [<:] (subtyping is defined as in Gay and Hole (2005); Carbone et al. (2007b)), [Inact], [NRes] are standard. Rule [CRes] assigns a type to process prefixed by a channel restriction based on the typing of the subprocess: subprocess P must be well-typed by the disjoint union of all end-point $T_1, ..., T_n$. Rules [Var] and [Def] are standard.

Type-checking Examples We type the programs of the Addition, Delegation and Session-sending examples given respectively in Sections 2.1, 2.4 and 2.6.3, with focus on the typing of the process under the session initiation prefix. We present also an example of multicast delegation and its typing. The typing of the session initiation is trivial, resulting in an `end` only Δ. For all processes of each example, the union of `end` only types is compatible and so, rule [Conc] is easily proven.

Addition. We write *client* as $\overline{a}_{[2,3,4]}(x_1, x_2, x_3).Q_1$, *addition* as $\overline{a}_{[2]}(x_1, x_2, x_3).Q_2$, *successor* as $\overline{a}_{[3]}(x_1, x_2, x_3).Q_3$ and *predecessor* as $\overline{a}_{[4]}(x_1, x_2, x_3).Q_4$. Type-checking is defined for processes under the session initiation prefix with typing context $\Gamma = \{a : \langle G \rangle\}$ where G is defined as in Section 2.1, letting $\mathtt{Client} = 1$, $\mathtt{Addition} = 2$, $\mathtt{Successor} = 3$, $\mathtt{Predecessor} = 4$:

$$\Gamma \vdash Q_1 \triangleright x_1, x_2, x_3 : 1!\langle int, int\rangle; 1\&\{answer : 1?\langle int\rangle; \mathsf{end}@\texttt{Client}$$
$$\Gamma \vdash Q_2 \triangleright x_1, x_2, x_3 : 1?\langle int, int\rangle; \mu t.2, 3\oplus$$
$$\{true : 1 \oplus \{answer : 1!\langle int\rangle; \mathsf{end},$$
$$false : 2!\langle int\rangle; 3!\langle int\rangle; 2?\langle int\rangle; 3?\langle int\rangle; t\}@\texttt{Addition}$$
$$\Gamma \vdash Q_3 \triangleright x_1, x_2, x_3 : \mu t.2\&\{true : \mathsf{end}, false : 2?\langle int\rangle; 2!\langle int\rangle; t\}@\texttt{Successor}$$
$$\Gamma \vdash Q_4 \triangleright x_1, x_2, x_3 : \mu t.3\&\{true : \mathsf{end}, false : 3?\langle int\rangle; 3!\langle int\rangle; t\}@\texttt{Predecessor}$$

Delegation. With the assumption list $\Gamma = \{a : \langle G_a\rangle, b : \langle G_b\rangle\}$ where G_a and G_b are:

$$G_b = \quad A \to D\colon 1\ \langle 1!\langle string\rangle; \mathsf{end}@\texttt{A}\rangle.\mathsf{end}$$
$$G_a = \quad A \to C\colon 1\ \langle string\rangle.B \to C\colon 2\ \langle string\rangle.\mathsf{end}.$$

letting $\texttt{A} = 1$, $\texttt{B} = 2$, $\texttt{C} = 3$, $\texttt{D} = 4$, the typechecking of the processes is defined as follows:

$$\Gamma \vdash A \triangleright y_1 : 1!\langle 1!\langle string\rangle; \mathsf{end}@\texttt{A}\rangle@\texttt{A}, x_1, x_2 : 1!\langle string\rangle@\texttt{A}$$
$$\Gamma \vdash B \triangleright x_1, x_2 : 2!\langle string\rangle@\texttt{B}$$
$$\Gamma \vdash C \triangleright x_1, x_2 : 1?\langle string\rangle; 2?\langle string\rangle@\texttt{C}$$
$$\Gamma \vdash D \triangleright y_1 : 1?\langle 1!\langle string\rangle; \mathsf{end}@\texttt{A}\rangle@\texttt{D}$$

Session-sending. Let $\texttt{p} = 1$ and $\texttt{p}' = 2$. The projection of the global type given in Section 2.6.3 onto participants $\texttt{p}$ and $\texttt{p}'$ is given below:

$$\mu\mathbf{t}.\mathrm{n}!\langle\langle\mathbf{t}\rangle\rangle; \mathsf{end}@\texttt{p}$$
$$\mu\mathbf{t}.\mathrm{n}?\langle\langle\mathbf{t}\rangle\rangle; \mathsf{end}@\texttt{p}'$$

According to the typing rules, the above types do not type-check the processes:

$$x!\langle a\rangle; \mathbf{0}$$
$$x?\langle y\rangle;$$

as the one-time unfolding of $\texttt{p}$'s end-point:

$$\mathrm{n}!\langle\langle\mu\mathbf{t}.\mathrm{n}!\langle\langle\mathbf{t}\rangle\rangle; \mathsf{end}\rangle\rangle; \mathsf{end}@\texttt{p}$$

returns the following type for a:

$$\mu\mathbf{t}.\mathrm{n}!\langle\langle\mathbf{t}\rangle\rangle; \mathsf{end}@\texttt{p}$$

The latter does not match the global type of a in the context of typing Γ.

Multicast delegation. Consider a different version of the delegation example where A sends both the title and author of the book to C on two different channels, and delegates in multicast such behaviour to B and D. These interactions are described in global types as:

$$\begin{aligned} G_b = &\quad \mathtt{A} \to \mathtt{B},\mathtt{D}\colon 1\ \langle 1!\langle string\rangle, 2!\langle string\rangle@\mathtt{A}\rangle.\mathsf{end} \\ G_a = &\quad \mathtt{A} \to \mathtt{C}\colon 1\ \langle string\rangle.\mathtt{A} \to \mathtt{C}\colon 2\ \langle string\rangle.\mathsf{end}. \end{aligned}$$

The definition of processes are given below:

$A \triangleq \overline{a}_{[2]}\,(x_1, x_2).\overline{b}_{[2,3]}\,(y_1, y_2).y_1, y_2!\langle x_1, x_2\rangle;$
$B \triangleq b_{[2]}\,(y_1, y_2).y_1?(y_3, y_4); y_3!\langle \text{“}\textit{The computer and the brain}\text{”}\rangle;$
$C \triangleq a_{[2]}\,(x_1, x_2).x_1?(y'); x_2?(y'');$
$D \triangleq b_{[3]}\,(y_1, y_2).y_2?(y_3, y_4); y_4!\langle \text{“}\textit{John von Neumann}\text{”}\rangle;$

and the typechecking of the processes is defined as follows:

$\Gamma \vdash A \triangleright y_1, y_2 : 1, 2!\langle 1!\langle string\rangle, 2!\langle string\rangle@\mathtt{A}\rangle@\mathtt{A}, x_1, x_2 : 1!\langle string\rangle; 2!\langle string\rangle@\mathtt{A}$
$\Gamma \vdash B \triangleright y_1, y_2 : 1?\langle 1!\langle string\rangle; \mathsf{end}@\mathtt{A}\rangle@\mathtt{B}$
$\Gamma \vdash C \triangleright x_1, x_2 : 1?\langle string\rangle; 2?\langle string\rangle@\mathtt{C}$
$\Gamma \vdash D \triangleright y_1, y_2 : 2?\langle 2!\langle string\rangle; \mathsf{end}@\mathtt{A}\rangle@\mathtt{D}$

2.6.5 Properties of the Type System

We prove three properties of this system: type preservation, communication-safety and session fidelity. Type preservation states that if a term is well-typed and it reduces to a new term, then the new term is also well-typed. Communication-safety is stated at the beginning of the introduction. Session fidelity states that the interactions of a typable process follow exactly the description of the global type. The definitions and proofs of this chapter are based on the definitions and proofs of Honda et al. (2008a)

Reduction over session typings is introduced below, representing interactions in processes. We assume well-formedness of types.

$$\mathrm{m}_1, ..., \mathrm{m}_n!\langle \tilde{V} \rangle; T@\mathtt{p}, \mathrm{m}_1?\langle \tilde{V} \rangle; T_1@\mathtt{p}_1, ..., \mathrm{m}_n?\langle \tilde{V} \rangle; T_n@\mathtt{p}_n \xrightarrow{\mathrm{m}_1,...,\mathrm{m}_n} T@\mathtt{p}, T_1@\mathtt{p}_1, ..., T_n@\mathtt{p}_n \quad \text{[TR-Mult]}$$

$$\mathrm{m}_1, ..., \mathrm{m}_n!\langle T'_1, ..., T'_n@\mathtt{p}' \rangle; T@\mathtt{p}, \mathrm{m}_1?\langle T'_1@\mathtt{p}' \rangle; T_1@\mathtt{p}_1, ..., \mathrm{m}_n?\langle T'_n@\mathtt{p}' \rangle; T_n@\mathtt{p}_n \xrightarrow{\mathrm{m}_1,...,\mathrm{m}_n} T@\mathtt{p}, T_1@\mathtt{p}_1, ..., T_n@\mathtt{p}_n \quad \text{[TR-MultD]}$$

$$\mathrm{m}_1, ..., \mathrm{m}_n \oplus \{..., \ l : T, \ ...\}@\mathtt{p}, \mathrm{m}_1\&\{..., l : T_1, \ ...\}@\mathtt{p}_1, ..., \mathrm{m}_n\&\{..., l : T_n, \ ...\}@\mathtt{p}_n \xrightarrow{\mathrm{m}_1,...,\mathrm{m}_n} T@\mathtt{p}, T_1@\mathtt{p}_1, ..., T_n@\mathtt{p}_n \quad \text{[TR-MultL]}$$

$$\frac{T_1@\mathtt{p}_1, ..., T_n@\mathtt{p}_n \xrightarrow{\tilde{\mathrm{m}}} T'_1@\mathtt{p}_1, ..., T'_n@\mathtt{p}_n}{\Delta, \tilde{\kappa}^{\mathtt{p}_1}, ..., \tilde{\kappa}^{\mathtt{p}_n} : T_1@\mathtt{p}_1, ..., T_n@\mathtt{p}_n, _ \xrightarrow{\kappa_{\tilde{\mathrm{m}}}} \Delta, \tilde{\kappa}^{\mathtt{p}_1}, ..., \tilde{\kappa}^{\mathtt{p}_n} : T'_1@\mathtt{p}_1, ..., T'_n@\mathtt{p}_n, _} \quad \text{[TR-Context]}$$

Rule [TR-Mult] represents the action of sending single or multiple values of type $\tilde{V}$ through channels $\tilde{\mathrm{m}}$ as in rule [R-Multicast]. Rule [TR-MultD] represents the action of delegating single or multiple behaviour of type $\tilde{T}'$ through channels $\tilde{\mathrm{m}}$, following the typing strategy of the type system. Rule [TR-MultL] represents the action of sending a label and is symmetric to rule [TR-Mult]. Rule [TR-Context] defines typing reduction in session typing through the fore-mentioned rules in end-point types.

We write $k_{\tilde{m}}$ for $\tilde{k}$ to explicitly declare the channels as in global types. The following notations are used throughout the proofs of the theorems.

Notation 2.6.20. "By inversion" denotes inversion on a rule. That is, a conclusion judgment that is achieved by applying a certain rule is true if the premises on that rule are true.

Notation 2.6.21. "By rule" denotes applying a rule. That is, given the premises and side conditions of a rule then we can conclude the judgment by applying that rule.

Notation 2.6.22. "By i.h." or "By induction" denotes induction hypothesis. That is, a subterm holds the property we are proving.

For the proof of type preservation, we need three standard properties: *channel replacement*, *weakening* and *substitution* lemma. We need the channel replacement lemma for rules [R-Link], [R-Multicast] and [R-ProcCall], weakening for rule [R-ProcCall] and subject congruence, and substitution for rule [R-Multicast] and [R-ProcCall].

Lemma 2.6.23 (substitution and weakening)**.** (1) $\Gamma, \tilde{x}:\tilde{V} \vdash P \triangleright \Delta$ and $\Gamma \vdash \tilde{v}:\tilde{V}$ imply $\Gamma \vdash P[\tilde{v}/\tilde{x}] \triangleright \Delta$. (2) Whenever $\Gamma \vdash P \triangleright \Delta$ is derivable then its weakening, $\Gamma \vdash P \triangleright \Delta, \Delta'$ for disjoint Δ' where Δ' contains only empty type contexts and for types end, is also derivable.

Proof. (1) is trivial by induction on the derivation of $\Gamma \vdash P \triangleright \Delta$. We present the most appealing cases, including those most difficult.

$(1)\Gamma, \tilde{x} : \tilde{V} \vdash \tilde{x} : \tilde{V}$ and $(2)\Gamma \vdash \tilde{v} : \tilde{V}$ — By assumption
$\Gamma \vdash \tilde{x}[\tilde{v}/\tilde{x}]: \tilde{V}$ — By assumption (2)
Note that $\tilde{x}[\tilde{v}/\tilde{x}] = \tilde{v}$.

$\Gamma, \tilde{y} : \tilde{V}', \tilde{x} : \tilde{V} \vdash \tilde{y} : \tilde{V}'$ and $\Gamma \vdash \tilde{v} : \tilde{V}$ — By assumption
$\Gamma, \tilde{y} : \tilde{V}' \vdash \tilde{y}[\tilde{v}/\tilde{x}]: \tilde{V}'$ — By rule
Note that $\tilde{y}[\tilde{v}/\tilde{x}] = \tilde{y}$.

$\Gamma, \tilde{x} : \tilde{V} \vdash \bar{u}[2..\mathrm{n}](\tilde{y}).P \triangleright \Delta$ and $\Gamma \vdash \tilde{v} : \tilde{V}$ — By assumption
$\Gamma, \tilde{x} : \tilde{V} \vdash u: \langle G \rangle \quad \Gamma, \tilde{x} : \tilde{V} \vdash P \triangleright \Delta, \tilde{y}: (G\restriction 1)@1 \quad |\tilde{y}| = |\mathsf{sid}(G)|$ and $\Gamma \vdash \tilde{v} : \tilde{V}$ — By inversion
$\Gamma \vdash u[\tilde{v}/\tilde{x}]: \langle G \rangle \quad \Gamma \vdash P[\tilde{v}/\tilde{x}] \triangleright \Delta, \tilde{y}: (G\restriction 1)@1 \quad |\tilde{y}| = |\mathsf{sid}(G)|$ — By induction
$\Gamma \vdash \bar{u}[\tilde{v}/\tilde{x}][2..\mathrm{n}](\tilde{y}).P[\tilde{v}/\tilde{x}] \triangleright \Delta$ — By rule [MCast]
Note that $\{\bar{u}[2..\mathrm{n}](\tilde{y}).P\}[\tilde{v}/\tilde{x}] = \bar{u}[\tilde{v}/\tilde{x}][2..\mathrm{n}](\tilde{y}).P[\tilde{v}/\tilde{x}]$.

Case [MAcc] is symmetric.

$\Gamma, \tilde{x} : \tilde{V} \vdash k_{\tilde{\mathrm{m}}}!\langle \tilde{e} \rangle; P \triangleright \Delta, \tilde{k}: \tilde{\mathrm{m}}!\langle \tilde{V}' \rangle; T@\mathtt{p}$ and $\Gamma \vdash \tilde{v} : \tilde{V}$ — By assumption
$\Gamma, \tilde{x} : \tilde{V} \vdash \tilde{e}: \tilde{V}' \qquad \Gamma, \tilde{x} : \tilde{V} \vdash P \triangleright \Delta, \tilde{k}: T@\mathtt{p}$ and $\Gamma \vdash \tilde{v} : \tilde{V}$ — By inversion
$\Gamma \vdash e[\tilde{v}/\tilde{x}]: \tilde{V}' \qquad \Gamma \vdash P[\tilde{v}/\tilde{x}] \triangleright \Delta, \tilde{k}: T@\mathtt{p}$ — By induction
$\Gamma \vdash k_{\tilde{\mathrm{m}}}!\langle \tilde{e}[\tilde{v}/\tilde{x}] \rangle; P[\tilde{v}/\tilde{x}] \triangleright \Delta, \tilde{k}: \tilde{\mathrm{m}}!\langle \tilde{V}' \rangle; T@\mathtt{p}$ — By rule [Send]
Note that $\{k_{\tilde{\mathrm{m}}}!\langle \tilde{e} \rangle; P\}[\tilde{v}/\tilde{x}] = k_{\tilde{\mathrm{m}}}!\langle \tilde{e}[\tilde{v}/\tilde{x}] \rangle; P[\tilde{v}/\tilde{x}]$.

$\Gamma, \tilde{x} : \tilde{V} \vdash k_{\mathrm{m}}?\langle \tilde{y} \rangle; P \triangleright \Delta, \tilde{k}: \mathrm{m}?\langle \tilde{V}' \rangle; T@\mathtt{p}$ and $\Gamma \vdash \tilde{v} : \tilde{V}$ — By assumption
$\Gamma, \tilde{y}: \tilde{V}', \tilde{x} : \tilde{V} \vdash P \triangleright \Delta, \tilde{k}: T@\mathtt{p}$ and $\Gamma \vdash \tilde{v} : \tilde{V}$ — By inversion
$\Gamma, \tilde{y}: \tilde{V}' \vdash P[\tilde{v}/\tilde{x}] \triangleright \Delta, \tilde{k}: T@\mathtt{p}$ — By induction
$\Gamma \vdash k_{\mathrm{m}}?\langle \tilde{y} \rangle; P[\tilde{v}/\tilde{x}] \triangleright \Delta, \tilde{k}: \mathrm{m}?\langle \tilde{V}' \rangle; T@\mathtt{p}$ — By rule [Recv]
Note that $\{k_{\mathrm{m}}?\langle \tilde{y} \rangle; P\}[\tilde{v}/\tilde{x}] = k_{\mathrm{m}}?\langle \tilde{y} \rangle; P[\tilde{v}/\tilde{x}]$.

Cases [Thr], [Cat], [Sel], [Br] are symmetric.

$\Gamma, \tilde{x} : \tilde{V} \vdash P \mid Q \triangleright \Delta \circ \Delta'$ and $\Gamma \vdash \tilde{v} : \tilde{V}$ By assumption
$\Gamma, \tilde{x} : \tilde{V} \vdash P \triangleright \Delta \quad \Gamma, \tilde{x} : \tilde{V} \vdash Q \triangleright \Delta' \quad \Delta \asymp \Delta'$ and $\Gamma \vdash \tilde{v} : \tilde{V}$ By inversion
$\Gamma \vdash P[\tilde{v}/\tilde{x}] \triangleright \Delta \quad \Gamma \vdash Q[\tilde{v}/\tilde{x}] \triangleright \Delta' \quad \Delta \asymp \Delta'$ By induction
$\Gamma \vdash P[\tilde{v}/\tilde{x}] \mid Q[\tilde{v}/\tilde{x}] \triangleright \Delta \circ \Delta'$ By rule [Conc]
Note that $\{P \mid Q\}[\tilde{v}/\tilde{x}] = P[\tilde{v}/\tilde{x}] \mid Q[\tilde{v}/\tilde{x}]$.

Cases [If], [<:], [Inact], [NRes], [CRes] are symmetric.

$\Gamma, X : \tilde{V}'\tilde{T}, \tilde{x} : \tilde{V} \vdash X\langle \tilde{e}, \tilde{k} \rangle \triangleright \Delta, \tilde{k} : \tilde{T}@\tilde{\mathtt{p}}$ and $\Gamma \vdash \tilde{v} : \tilde{V}$ By assumption
$\Gamma, \tilde{x} : \tilde{V} \vdash \tilde{e} : \tilde{V}' \quad \Delta$ end only and $\Gamma \vdash \tilde{v} : \tilde{V}$ By inversion
$\Gamma \vdash \tilde{e}[\tilde{v}/\tilde{x}] : \tilde{V}' \quad \Delta$ end only By induction
$\Gamma, X : \tilde{V}'\tilde{T} \vdash X\langle \tilde{e}[\tilde{v}/\tilde{x}], \tilde{k} \rangle \triangleright \Delta, \tilde{k} : \tilde{T}@\tilde{\mathtt{p}}$ By rule [Var]
Note that $\{X\langle \tilde{e}, \tilde{k} \rangle\}[\tilde{v}/\tilde{x}] = X\langle \tilde{e}[\tilde{v}/\tilde{x}], \tilde{k} \rangle$.

$\Gamma, \tilde{x} : \tilde{V} \vdash \mathsf{def}\ X(\tilde{y}, \tilde{y'}) = P\ \mathsf{in}\ Q \triangleright \Delta$ and $\Gamma \vdash \tilde{v} : \tilde{V}$ By assumption
$\Gamma, X : \tilde{V}'\tilde{T}, \tilde{y} : \tilde{V}', \tilde{x} : \tilde{V} \vdash P \triangleright \tilde{y'} : \tilde{T}@\tilde{\mathtt{p}} \quad \Gamma, X : \tilde{V}'\tilde{T}, \tilde{x} : \tilde{V} \vdash Q \triangleright \Delta$
and $\Gamma \vdash \tilde{v} : \tilde{V}$ By inversion
$\Gamma, X : \tilde{V}'\tilde{T}, \tilde{y} : \tilde{V}' \vdash P[\tilde{v}/\tilde{x}] \triangleright \tilde{y'} : \tilde{T}@\tilde{\mathtt{p}} \quad \Gamma, X : \tilde{V}'\tilde{T} \vdash Q[\tilde{v}/\tilde{x}] \triangleright \Delta$ By induction
$\Gamma \vdash \mathsf{def}\ X(\tilde{y}, \tilde{y'}) = P[\tilde{v}/\tilde{x}]\ \mathsf{in}\ Q[\tilde{v}/\tilde{x}] \triangleright \Delta$ By rule [Def]
Note that $\{\mathsf{def}\ X(\tilde{y}, \tilde{y'}) = P\ \mathsf{in}\ Q\}[\tilde{v}/\tilde{x}] = \mathsf{def}\ X(\tilde{y}, \tilde{y'}) = P[\tilde{v}/\tilde{x}]\ \mathsf{in}\ Q[\tilde{v}/\tilde{x}]$.

(2) is trivial by induction on the derivation of $\Gamma \vdash P \triangleright \Delta$. We present the most appealing cases, including those most difficult.

$\Gamma \vdash \bar{u}[2..\mathrm{n}](\tilde{y}).P \triangleright \Delta$ and $\Delta' = \mathsf{end}$ *only* By assumption
$\Gamma \vdash u : \langle G \rangle \quad \Gamma \vdash P \triangleright \Delta, \tilde{y} : (G\restriction 1)@1 \quad |\tilde{y}| = |\mathsf{sid}(G)|$ and $\Delta' = \mathsf{end}$ *only* By inversion
$\Gamma \vdash u : \langle G \rangle \quad \Gamma \vdash P \triangleright \Delta, \tilde{y} : (G\restriction 1)@1, \Delta' \quad |\tilde{y}| = |\mathsf{sid}(G)|$ By induction
Note that $\Delta, \tilde{y} : (G\restriction 1)@1, \Delta' = \Delta, \Delta', \tilde{y} : (G\restriction 1)@1$
$\Gamma \vdash \bar{u}[2..\mathrm{n}](\tilde{y}).P \triangleright \Delta, \Delta'$ By rule [MCast]

Case [MAcc] is symmetric.

$\Gamma \vdash k_{\tilde{\mathrm{m}}}!\langle \tilde{e} \rangle; P \triangleright \Delta, \tilde{k} : \tilde{\mathrm{m}}!\langle \tilde{V}' \rangle; T@\mathtt{p}$ and $\Delta' = \mathsf{end}$ *only* By assumption
$\Gamma \vdash \tilde{e} : \tilde{V}' \quad \Gamma \vdash P \triangleright \Delta, \tilde{k} : T@\mathtt{p}$ and $\Delta' = \mathsf{end}$ *only* By inversion
$\Gamma \vdash e : \tilde{V}' \quad \Gamma \vdash P \triangleright \Delta, \tilde{k} : T@\mathtt{p}, \Delta'$ By induction

Note that $\Delta, \tilde{k}\colon T@\mathtt{p}, \Delta' = \Delta, \Delta', \tilde{k}\colon T@\mathtt{p}$
$\Gamma \vdash k_{\tilde{\mathrm{m}}}!\langle \tilde{e}\rangle; P \triangleright \Delta, \Delta', \tilde{k}\colon \tilde{\mathrm{m}}!\langle \tilde{V}'\rangle; T@\mathtt{p}$ By rule [Send]
Note that $\Delta, \Delta', \tilde{k}\colon \tilde{\mathrm{m}}!\langle \tilde{V}'\rangle; T@\mathtt{p} = \Delta, \tilde{k}\colon \tilde{\mathrm{m}}!\langle \tilde{V}'\rangle; T@\mathtt{p}, \Delta'$

$\Gamma \vdash k_{\mathrm{m}}?\langle \tilde{y}\rangle; P \triangleright \Delta, \tilde{k}\colon \mathrm{m}?\langle \tilde{V}'\rangle; T@\mathtt{p}$ and $\Delta' = \mathsf{end}$ *only* By assumption
$\Gamma, \tilde{y}\colon \tilde{V}' \vdash P \triangleright \Delta, \tilde{k}\colon T@\mathtt{p}$ and $\Delta' = \mathsf{end}$ *only* By inversion
$\Gamma, \tilde{y}\colon \tilde{V}' \vdash P \triangleright \Delta, \tilde{k}\colon T@\mathtt{p}, \Delta'$ By induction
Note that $\Delta, \tilde{k}\colon T@\mathtt{p}, \Delta' = \Delta, \Delta', \tilde{k}\colon T@\mathtt{p}$
$\Gamma \vdash k_{\mathrm{m}}?\langle \tilde{y}\rangle; P \triangleright \Delta, \Delta', \tilde{k}\colon \mathrm{m}?\langle \tilde{V}'\rangle; T@\mathtt{p}$ By rule [Recv]
Note that $\Delta, \Delta', \tilde{k}\colon \mathrm{m}?\langle \tilde{V}'\rangle; T@\mathtt{p} = \Delta, \tilde{k}\colon \mathrm{m}?\langle \tilde{V}'\rangle; T@\mathtt{p}, \Delta'$.

Cases [Thr], [Cat], [Sel], [Br] are symmetric.

$\Gamma \vdash P \mid Q \triangleright \Delta \circ \Delta''$ and $\Delta' = \mathsf{end}$ *only* By assumption
$\Gamma \vdash P \triangleright \Delta \quad \Gamma \vdash Q \triangleright \Delta'' \quad \Delta \asymp \Delta''$ and $\Delta' = \mathsf{end}$ *only* By inversion
$\Gamma \vdash P \triangleright \Delta, \Delta' \quad \Gamma \vdash Q \triangleright \Delta'', \Delta' \quad \Delta \asymp \Delta''$ By induction
Note that $\Delta, \Delta' \asymp \Delta'', \Delta'$
$\Gamma \vdash P \mid Q \triangleright \Delta, \Delta' \circ \Delta'', \Delta'$ By rule [Conc]
Note that $\Delta, \Delta' \circ \Delta'', \Delta' = \Delta \circ \Delta'', \Delta'$.

Cases [If], [<:], [Inact], [NRes], [CRes] are symmetric.

$\Gamma, X\colon \tilde{V}'\tilde{T} \vdash X\langle \tilde{e}, \tilde{k}\rangle \triangleright \Delta, \tilde{k} : \tilde{T}@\tilde{\mathtt{p}}$ and $\Delta' = \mathsf{end}$ *only* By assumption
$\Gamma \vdash \tilde{e}\colon \tilde{V}' \quad \Delta$ end *only* and $\Delta' = \mathsf{end}$ *only* By inversion
$\Gamma \vdash \tilde{e}\colon \tilde{V}' \quad \Delta, \Delta'$ end *only* By induction
$\Gamma, X\colon \tilde{V}'\tilde{T} \vdash X\langle \tilde{e}, \tilde{k}\rangle \triangleright \Delta, \Delta', \tilde{k} : \tilde{T}@\tilde{\mathtt{p}}$ By rule [Var]
Note that $\Delta, \Delta', \tilde{k} : \tilde{T}@\tilde{\mathtt{p}} = \Delta, \tilde{k} : \tilde{T}@\tilde{\mathtt{p}}, \Delta'$.

$\Gamma \vdash \mathsf{def}\ X(\tilde{y}, \tilde{y'}) = P\ \mathsf{in}\ Q \triangleright \Delta$ and $\Delta' = \mathsf{end}$ *only* By assumption
$\Gamma, X\colon \tilde{V}'\tilde{T}, \tilde{y} : \tilde{V}' \vdash P \triangleright \tilde{y'} : \tilde{T}@\tilde{\mathtt{p}} \quad \Gamma, X\colon \tilde{V}'\tilde{T} \vdash Q \triangleright \Delta$ and $\Delta' = \mathsf{end}$ *only* By inversion
$\Gamma, X\colon \tilde{V}'\tilde{T}, \tilde{y} : \tilde{V}' \vdash P \triangleright \tilde{y'} : \tilde{T}@\tilde{\mathtt{p}} \quad \Gamma, X\colon \tilde{V}'\tilde{T} \vdash Q \triangleright \Delta, \Delta'$ By induction
$\Gamma \vdash \mathsf{def}\ X(\tilde{y}, \tilde{y'}) = P\ \mathsf{in}\ Q \triangleright \Delta, \Delta'$ By rule [Def]

□

Lemma 2.6.24 (Channel Replacement)**.** If $\Gamma \vdash P \triangleright \Delta, \tilde{x}\colon T@\mathtt{p}$ and $\tilde{\kappa}^{\mathtt{p}} \notin dom(\Delta)$, then $\Gamma \vdash P[\tilde{\kappa}^{\mathtt{p}}/\tilde{x}] \triangleright \Delta, \tilde{\kappa}^{\mathtt{p}}\colon T@\mathtt{p}$.

Proof. A straightforward induction on the derivation tree for P. We give the proof for the most interesting cases, including the most difficult.
Case: [Conc]

$$\frac{\Gamma \vdash P \triangleright \Delta_1 \qquad \Gamma \vdash Q \triangleright \Delta_2 \qquad \Delta_1 \asymp \Delta_2}{\Gamma \vdash P \mid Q \triangleright \Delta, \tilde{x}\colon T@\mathtt{p}}$$

$\Gamma \vdash P \mid Q \triangleright \Delta, \tilde{x}\colon T@\mathtt{p}$ and $\tilde{\kappa}^{\mathtt{p}} \notin dom(\Delta)$	By assumption
$\Gamma \vdash P \triangleright \Delta_1 \quad \Gamma \vdash Q \triangleright \Delta_2 \quad \Delta_1 \asymp \Delta_2$	
where $\Delta, \tilde{x}\colon T@\mathtt{p} = \Delta_1 \circ \Delta_2$ and $\tilde{\kappa}^{\mathtt{p}} \notin dom(\Delta)$	By inversion on [Conc]
First Subcase: $\tilde{x} : T@\mathtt{p} \in \Delta_1$ and $\tilde{x} : T@\mathtt{p} \notin \Delta_2$	By $\Delta_1 \asymp \Delta_2$
$\Delta_1 = \Delta_1', \tilde{x}\colon T@\mathtt{p} \quad \Gamma \vdash P[\tilde{\kappa}^{\mathtt{p}}/\tilde{x}] \triangleright \Delta_1', \tilde{\kappa}^{\mathtt{p}}\colon T@\mathtt{p}$	By induction
$\Gamma \vdash P[\tilde{\kappa}^{\mathtt{p}}/\tilde{x}] \mid Q \triangleright \Delta_1', \tilde{\kappa}^{\mathtt{p}}\colon T@\mathtt{p} \circ \Delta_2$	By rule [Conc]
$\Delta_1', \tilde{\kappa}^{\mathtt{p}}\colon T@\mathtt{p} \circ \Delta_2 = \Delta_1' \circ \Delta_2, \tilde{\kappa}^{\mathtt{p}}\colon T@\mathtt{p}$	$\tilde{\kappa}^{\mathtt{p}} \notin dom((\Delta))$
$\Gamma \vdash P[\tilde{\kappa}^{\mathtt{p}}/\tilde{x}] \mid Q \triangleright \Delta, \tilde{\kappa}^{\mathtt{p}}\colon T@\mathtt{p}$	
Second Subcase: $\tilde{x} : T@\mathtt{p} \in \Delta_2$ and $\tilde{x} : T@\mathtt{p} \notin \Delta_1$	By $\Delta_1 \asymp \Delta_2$
$\Delta_2 = \Delta_2', \tilde{x}\colon T@\mathtt{p} \quad \Gamma \vdash Q[\tilde{\kappa}^{\mathtt{p}}/\tilde{x}] \triangleright \Delta_2', \tilde{\kappa}^{\mathtt{p}}\colon T@\mathtt{p}$	By induction
$\Gamma \vdash P \mid Q[\tilde{\kappa}^{\mathtt{p}}/\tilde{x}] \triangleright \Delta_1 \circ (\Delta_2', \tilde{\kappa}^{\mathtt{p}}\colon T@\mathtt{p})$	By rule [Conc]
$(\Delta_2', \tilde{\kappa}^{\mathtt{p}}\colon T@\mathtt{p}) = \Delta_1' \circ \Delta_2, \tilde{\kappa}^{\mathtt{p}}\colon T@\mathtt{p}$	$\tilde{\kappa}^{\mathtt{p}} \notin dom((\Delta))$
$\Gamma \vdash P \mid Q[\tilde{\kappa}^{\mathtt{p}}/\tilde{x}] \triangleright \Delta, \tilde{\kappa}^{\mathtt{p}}\colon T@\mathtt{p}$	

Note that $(P \mid Q)[\tilde{\kappa}^{\mathtt{p}}/\tilde{x}] = P[\tilde{\kappa}^{\mathtt{p}}/\tilde{x}] \mid Q$ or $(P \mid Q)[\tilde{\kappa}^{\mathtt{p}}/\tilde{x}] = P \mid Q[\tilde{\kappa}^{\mathtt{p}}/\tilde{x}]$.

Case: [If]

$$\frac{\Gamma \vdash e \triangleright \mathsf{bool} \qquad \Gamma \vdash P \triangleright \Delta, \tilde{x} : T@\mathtt{p} \qquad \Gamma \vdash Q \triangleright \Delta, \tilde{x} : T@\mathtt{p}}{\Gamma \vdash \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \triangleright \Delta, \tilde{x} : T@\mathtt{p}}$$

$\Gamma \vdash \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \triangleright \Delta, \tilde{x} : T@\mathtt{p}$ and $\tilde{\kappa}^{\mathtt{p}} \notin dom((\Delta))$	By assumption
$\Gamma \vdash P \triangleright \Delta, \tilde{x} : T@\mathtt{p}$ and $\Gamma \vdash Q \triangleright \Delta, \tilde{x} : T@\mathtt{p}$ and $\tilde{\kappa}^{\mathtt{p}} \notin dom(\Delta)$	By inversion on [If]
$\Gamma \vdash P[\tilde{\kappa}^{\mathtt{p}}/\tilde{x}] \triangleright \Delta, \tilde{\kappa}^{\mathtt{p}} : T@\mathtt{p}$ and $\Gamma \vdash Q[\tilde{\kappa}^{\mathtt{p}}/\tilde{x}] \triangleright \Delta, \tilde{\kappa}^{\mathtt{p}} : T@\mathtt{p}$	By induction
$\Gamma \vdash (\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q)[\tilde{\kappa}^{\mathtt{p}}/\tilde{x}] \triangleright \Delta, \tilde{\kappa}^{\mathtt{p}} : T@\mathtt{p}$	By rule [If]

Note that $(\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q)[\tilde{\kappa}^{\mathtt{p}}/\tilde{x}] = \mathtt{if}\ e\ \mathtt{then}\ P[\tilde{\kappa}^{\mathtt{p}}/\tilde{x}]\ \mathtt{else}\ Q[\tilde{\kappa}^{\mathtt{p}}/\tilde{x}]$.

□

Definition 2.6.25 (prefix ordering)**.** Write $\tau, \tau', ..$ for prefixes occurring in an end-point type, say T (but not in its carried types), seen as nodes of T as a regular tree. We write $\tau \in T$ when τ occurs in T. Then we write $\tau_1 \prec \tau_2 \in T$ when τ_1 directly or indirectly prefixes τ_2 in T. Formally $\prec$ is the least partial order generated by:

$$\begin{array}{ll} \tau_1 \prec \tau_2 \in \mathrm{m}!/?\langle U\rangle; T' & \text{if } \tau_1 = \mathrm{m}!/?, \tau_2 \in T' \\ \tau_1 \prec \tau_2 \in \mathrm{m}\&/\oplus\{l_i : T_i\}_{i\in J} & \text{if } \tau_1 = \mathrm{m}\&/\oplus\ \exists i \in J.\, \tau_2 \in T_i \end{array}$$

Further we set $\tau_1 \prec \tau_2 \in T$ if $\tau_1 \prec \tau_2 \in T'$ and T' occurs in T but not in its carried types.

Definition 2.6.26. (1) (coherence of typings) We say Δ is coherent if $\Delta(\tilde{k})$ is coherent for each $\tilde{k} \in dom(\Delta)$.

(2) (full projection) Assume G is coherent and let $G \upharpoonright \mathsf{p}_i = T_i$ for each $\mathsf{p}_i \in \mathsf{pid}(G)$. Then $[\![G]\!]$, called *full projection of* P, denotes the family $\{T_i@\mathsf{p}_i\}$.

(3) (causal edges on $[\![G]\!]$) Given $[\![G]\!]$, regarding each type in $[\![G]\!]$ as the corresponding regular tree, we define the causal edges $\prec_{\mathtt{II}}$, $\prec_{\mathtt{IO}}$, $\prec_{\mathtt{OI}}$ and $\prec_{\mathtt{OO}}$ among its prefixes as:

$$\begin{array}{ll} \tau_1 \prec_{\mathtt{II}} \tau_2 & \text{if } \tau_1 \prec \tau_2 \text{ and } \tau_i = \mathrm{m}_i?/\&(i = 1, 2) \\ \tau_1 \prec_{\mathtt{IO}} \tau_2 & \text{if } \tau_1 \prec \tau_2,\ \phi_1 = \mathrm{m}_1?/\& \text{ and } \phi_2 = \mathrm{m}_2!/\oplus. \\ \tau_1 \prec_{\mathtt{OI}} \tau_2 & \text{if } \tau_1 \prec \tau_2,\ \phi_1 = \mathrm{m}_1!/\oplus \text{ and } \phi_2 = \mathrm{m}_2?/\&. \\ \tau_1 \prec_{\mathtt{OO}} \tau_2 & \text{if } \tau_1 \prec \tau_2,\ \tau_i = \mathrm{m}_i!/\oplus\ (i = 1, 2) \end{array}$$

Definition 2.6.27. $G \xrightarrow{\tilde{\mathrm{m}}} G'$ is defined by taking off a prefix at $\tilde{\mathrm{m}}$ in G not suppressed by $\prec_{\mathsf{II}}$, $\prec_{\mathsf{IO}}$, $\prec_{\mathsf{OI}}$ or $\prec_{\mathsf{OO}}$, to obtain G'.

Definition 2.6.28. We define $G \xrightarrow{\tilde{\mathrm{m}}} G'$ if $[\![G]\!] \xrightarrow{\tilde{\mathrm{m}}} [\![G']\!]$.

Definition 2.6.29. Preservation of a causal edge in a global type G onto the projection $[\![G]\!]$ is defined on the end-point type of the participant which appears in both prefixes of the causal edge.

Proposition 2.6.30. Each causal edge in G is preserved through the projection onto $[\![G]\!]$.

Proof. This is because projection preserves the action of the participant which appears in both prefixes of the causal edge and defines the causal edge. The statement becomes more clear when considering the four possible cases of edges on prefixes:

Case II. $B \to A\colon \mathrm{m}_1.G.C \to A\colon \mathrm{m}_2$, where $B \to A\colon \mathrm{m}_1 \prec_{\mathtt{II}} C \to A\colon \mathrm{m}_2$, is mapped into $\{(\mathrm{m}_1!; G \upharpoonright \mathtt{B})@\mathtt{B}, (\mathrm{m}_1?; G \upharpoonright \mathtt{A}; \mathrm{m}_2?)@\mathtt{A}, (G \upharpoonright \mathtt{C}; \mathrm{m}_2!)@\mathtt{C}\}$

Case IO. $B \to A\colon \mathrm{m}_1.G.A \to C\colon \mathrm{m}_2$, where $B \to A\colon \mathrm{m}_1 \prec_{\mathtt{IO}} A \to C\colon \mathrm{m}_2$, is mapped into $\{(\mathrm{m}_1!; G \upharpoonright \mathtt{B})@\mathtt{B}, (\mathrm{m}_1!; G \upharpoonright \mathtt{A}; \mathrm{m}_2!)@\mathtt{A}, (G \upharpoonright \mathtt{C}; \mathrm{m}_2?\langle\rangle)@\mathtt{C}\}$

Case OI. $A \to B\colon \mathrm{m}_1.G.C \to A\colon \mathrm{m}_2$, where $A \to B\colon \mathrm{m}_1 \prec_{\mathtt{OI}} C \to A\colon \mathrm{m}_2$, is mapped into $\{(\mathrm{m}_1?; G \upharpoonright \mathtt{B})@\mathtt{B}, (\mathrm{m}_1!; G \upharpoonright \mathtt{A}; \mathrm{m}_2?)@\mathtt{A}, (G \upharpoonright \mathtt{C}; \mathrm{m}_2!)@\mathtt{C}\}$

Case OO. $A \to B\colon \mathrm{m}_1.G.A \to C\colon \mathrm{m}_2$, where $A \to B\colon \mathrm{m}_1 \prec_{\mathtt{OO}} A \to C\colon \mathrm{m}_2$, is mapped into $\{(\mathrm{m}_1?; G \upharpoonright \mathtt{B})@\mathtt{B}, (\mathrm{m}_1!; G \upharpoonright \mathtt{A}; \mathrm{m}_2!)@\mathtt{A}, (G \upharpoonright \mathtt{C}; \mathrm{m}_2?)@\mathtt{C}\}$

□

Definition 2.6.31 (merge set)**.** Assume G is coherent. Then we say two prefixes ϕ_1, ϕ_2, s.t. $\phi_1 \in G_1$ and $\phi_2 \in G_2$ where $\mathtt{p} \to \mathtt{p}' : \mathrm{m}\{l_i : G_i\}_{[1..n]} \in G$ are *mergeable* with each other when they are collapsed in its projection up to the $\bowtie$ operator. A prefix is always mergeable with itself. Given a prefix ϕ, its *merge set* is the set of prefixes mergeable with ϕ.

Proposition 2.6.32. Two prefixes $\phi_1, \phi_2 = \mathtt{p} \to \mathtt{p}' : \mathrm{m}$ in G are mergeable iff there exists $T@\mathtt{p}, T'@\mathtt{p}' \in [\![G]\!]$ through projection such that $\mathrm{m}!/\oplus$ is an output prefix in T and $\mathrm{m}?/\&$ is an input prefix in T'.

Proof. This is because, in the defining clauses of projection, there are no other cases than the one for branching which collapse two prefixes up to the $\bowtie$ operator. □

Corollary 2.6.33. There is a bijection between the merge sets in G and the set of input prefixes in $[\![G]\!]$. Similarly for resp. the set of output prefixes in $[\![G]\!]$.

Proof. We take projection as the bijection function. From Proposition 2.6.32, we have that for each input prefix in $[\![G]\!]$ corresponds only one merge set and for each merge set corresponds only one input prefix. This is also valid for output prefixes. □

Definition 2.6.34. If a merge set of a prefix $\phi = \mathtt{p} \to \mathtt{p}' : \mathrm{m}$ of G is related to an input prefix $\mathrm{m}?/\&$ and an output prefix $\mathrm{m}!/\oplus$ in $[\![G]\!]$ (by Corollary 2.6.33), we say the former is the *projection preimage* or simply *preimage* of the latter, or the latter is the *projection image* or *image* of the former.

Proposition 2.6.35. (1) If a n-pair of prefixes in $[\![G]\!]$ forms a redex with respect to $\to$ then they are not prefixed by any pair of prefixes that form a $\prec_{\mathtt{II}}$, $\prec_{\mathtt{IO}}$, $\prec_{\mathtt{OI}}$ or $\prec_{\mathtt{OO}}$ dependency.

(2) Given G coherent, consider G' as the result of taking off the merge set of a prefix from G which is not suppressed by any of $\prec_{\mathtt{II}}$, $\prec_{\mathtt{IO}}$, $\prec_{\mathtt{OI}}$ or $\prec_{\mathtt{OO}}$ dependencies. Then G' is coherent.

(3) Let G be coherent. Then the causal edges between the two merge sets in G and their images in $[\![G]\!]$ are preserved. Further each redex pair in $[\![G]\!]$ is the image of a prefix in G.

Proof. For (1), observe that prefixes in rules [TR-Mult], [TR-MultD] and [TR-MultL] are in the minimal positions and since there is no permuation of prefixes, we conclude.

For (2), G' is coherent if it is linear and projection is well-defined for each participant of G'. For linearity, consider prefixes $\phi_{1,2}$ in G' sharing a channel. Then the prefixes are also in G. Since the prefix removed is not suppressed by any dependency then it is not part of any input or output chain. Thus, the causal edges between $\phi_{1,2}$ in G' are the same as in G. For projection is immediate since the prefixes of the merge set are the ones that collapse by projection in every branch, and so they are removed in each branch.

For (3), the first part is immediate from the construction. By Proposition 2.6.30, we have that every causal edges in G is preserved in $[\![G]\!]$ and by Definition 2.6.34, we have that each merge set of a prefix in G is related to a common input and output prefix in $[\![G]\!]$. For the second part assume that there is a n-pair redex in $[\![G]\!]$ whose elements have different preimages. Then we have co-occurring prefixes in G which are not related by dependencies. Since the co-occuring prefixes use the same channel, then there should be an input and output chain between them as G is coherent. Hence, we have a contradiction. □

Lemma 2.6.36. (1) $\Delta_1 \to^{\tilde{\mathtt{m}}} \Delta_1'$ and $\Delta_1 \asymp \Delta_2$ imply $\Delta_1' \asymp \Delta_2$ and $\Delta_1 \circ \Delta_2 \to^{\tilde{\mathtt{m}}} \Delta_1' \circ \Delta_2$.

(2) Let Δ be coherent. Then $\Delta \to \Delta'$ implies Δ' is coherent.

Proof. For (1) suppose $\Delta_1 \to \Delta_1'$ and $\Delta_1 \asymp \Delta_2$. Note $\Delta_1 \asymp \Delta_2$ means that each pair of vectors of channels from $\Delta_{1,2}$ either coincide or are disjoint, and that, if they coincide, their image are composable by $\circ$. Since no typed reduction rule invalidate either condition we conclude $\Delta_1' \asymp \Delta_2$. $\Delta_1 \circ \Delta_2 \to \Delta_1' \circ \Delta_2$ follows directly from [TR-Context].

For (2), suppose Δ is coherent and $\Delta \to \Delta'$. Suppose the associated redex is in $\Delta(\tilde{k})$. By coherence we can write $\Delta(\tilde{k})$ as $[\![G]\!]$ for some coherent G. Now consider the preimage of the associated redex in $[\![G]\!]$, whose existence is guaranteed by Proposition 2.6.35 (3). This preimage is not suppressed (related) by causal edges by Proposition 2.6.35 (1,3).

Reducing $[\![G]\!]$ corresponds to eliminating its preimage from G, say G', whose projection $[\![G']\!]$ precisely gives the result of reducing $[\![G]\!]$. Since G' is coherent by Proposition 2.6.35 (2) we are done. □

We need subject congruence when proving subject reduction for [Str].

Theorem 2.6.37 (type congruence)**.** $\Gamma \vdash P \triangleright \Delta$ and $P \equiv P'$ imply $\Gamma \vdash P' \triangleright \Delta$.

Proof. By rule induction on the derivation of $\Gamma \vdash P \triangleright \Delta$ when assuming that $P \equiv P'$ and $\Gamma \vdash P \triangleright \Delta$. For each structural congruence axiom, we consider each session type system rule that can generate $\Gamma \vdash P \triangleright \Delta$.

Case: $P \mid \mathbf{0} \equiv P$

$\Gamma \vdash P \mid \mathbf{0} \triangleright \Delta$	By assumption
$\Gamma \vdash P \triangleright \Delta_1$ and $\Gamma \vdash \mathbf{0} \triangleright \Delta_2$ where $\Delta = \Delta_1 \circ \Delta_2, \Delta_1 \asymp \Delta_2$	By inversion
Δ_2 is only end and for Δ_2 such that $dom(\Delta_1) \cap dom(\Delta_2) = \emptyset$	By inversion
then $\Gamma \vdash P \triangleright \Delta_1, \Delta_2$	By weakening
and $\Delta = \Delta_1 \circ \Delta_2 = \Delta_1, \Delta_2$	

$\Gamma \vdash P \triangleright \Delta$	By assumption
$\Gamma \vdash \mathbf{0} \triangleright \Delta'$ where Δ' is only end and $dom(\Delta) \cap dom(\Delta') = \emptyset$	By rule [Inact]
$\Gamma \vdash P \mid \mathbf{0} \triangleright \Delta, \Delta'$	By rule [Conc]
for $\Delta' = \emptyset$ we have that	
$\Gamma \vdash P \mid \mathbf{0} \triangleright \Delta$	

Case: $P \mid Q \equiv Q \mid P$

$\Gamma \vdash P \mid Q \triangleright \Delta$	By assumption
$\Gamma \vdash P \triangleright \Delta_1$ and $\Gamma \vdash Q \triangleright \Delta_2$	
where $\Delta = \Delta_1 \circ \Delta_2$ and $\Delta_1 \asymp \Delta_2 = \Delta_2 \asymp \Delta_1$	By inversion
$\Gamma \vdash Q \mid P \triangleright \Delta$	By rule [Conc]

The other case is symmetric to the above one.

Case: $(P \mid Q) \mid R \equiv P \mid (Q \mid R)$

$\Gamma \vdash (P \mid Q) \mid R \triangleright \Delta$ By assumption
$\Gamma \vdash P \triangleright \Delta_1$, $\Gamma \vdash Q \triangleright \Delta_2$ and $\Gamma \vdash R \triangleright \Delta_3$ where $\Delta = \Delta_1 \circ \Delta_2 \circ \Delta 3$
and $\Delta_1 \asymp \Delta_2 \asymp \Delta_3$ By inversion
$\Gamma \vdash P \mid (Q \mid R) \triangleright \Delta$ By rule [Conc]

The other case is symmetric to the above one. The other axioms are trivial. □

Theorem 2.6.38 (Type preservation)**.** $\Gamma \vdash P \triangleright \Delta$ with Δ coherent and $P \to P'$ imply $\Gamma \vdash P' \triangleright \Delta'$ where $\Delta = \Delta'$ or $\Delta \to \Delta'$ with Δ' coherent.

Proof. By rule induction on the derivation of $P \to P'$. There is a case for each operational semantics rule. For each operational semantics rule, we consider each type system rule that can generate $\Gamma \vdash P \triangleright \Delta$. By Lemma 2.6.36(2) we have that Δ' is coherent as well.

Case: [R-Link]

$$\overline{a}_{[2..n]}(\tilde{x}).P_1 \mid a_{[2]}(\tilde{x}).P_2 \mid \cdots \mid a_{[n]}(\tilde{x}).P_n \;\to\; (\nu\tilde{\kappa})(P_1[\tilde{\kappa}^1/\tilde{x}] \mid P_2[\tilde{\kappa}^2/\tilde{x}] \mid ... \mid P_n[\tilde{\kappa}^n/\tilde{x}])$$

$\Gamma \vdash \overline{a}_{[2..n]}(\tilde{x}).P_1 \mid a_{[2]}(\tilde{x}).P_2 \mid \cdots \mid a_{[n]}(\tilde{x}).P_n \triangleright \Delta$ By assumption
$\Gamma \vdash \overline{a}_{[2..n]}(\tilde{x}).P_1 \triangleright \Delta_1$ ... $\Gamma \vdash a_{[n]}(\tilde{x}).P_n \triangleright \Delta_n$ where $\Delta = \Delta_1 \circ ... \circ \Delta_n$
and $\Delta_1 \asymp ... \asymp \Delta_n$ By inversion on [Conc]

$\Gamma \vdash P_1 \triangleright \Delta_1, \tilde{x} : (G \upharpoonright 1)@1 \quad |\tilde{x}| = \mathsf{sid}(G)$ By inversion on [Mcast]
$\Gamma \vdash P_2 \triangleright \Delta_2, \tilde{x} : (G \upharpoonright 2)@2, |\tilde{x}| = \mathsf{sid}(G)$ By inversion on [Macc]
...
$\Gamma \vdash P_n \triangleright \Delta_n, \tilde{x} : (G \upharpoonright n)@n, |\tilde{x}| = \mathsf{sid}(G)$ By inversion on [Macc]

$\tilde{\kappa}^1 \notin dom(\Delta_1), ..., \tilde{\kappa}^n \notin dom(\Delta_n)$ for every $i \in \{1, ..., n\}.\tilde{\kappa}^i$ are newly generated

$\Gamma \vdash P_1[\tilde{\kappa}^1/\tilde{x}] \triangleright \Delta_1, \tilde{\kappa}^1 : (G \upharpoonright 1)@1$ By Lemma 2.6.24

...
$\Gamma \vdash P_n[\tilde{\kappa}^n/\tilde{x}] \triangleright \Delta_n, \tilde{\kappa}^n : (G \upharpoonright \mathtt{n})@\mathtt{n}$ By Lemma 2.6.24

$\tilde{\kappa}^1 : (G \upharpoonright \mathtt{1})@\mathtt{1} \asymp ... \asymp \tilde{\kappa}^n : (G \upharpoonright \mathtt{n})@\mathtt{n}$
$\Gamma \vdash P_1[\tilde{\kappa}^1/\tilde{x}] \mid P_2[\tilde{\kappa}^2/\tilde{x}] \mid ... \mid P_\mathtt{n}[\tilde{\kappa}^n/\tilde{x}] \triangleright$
$\Delta_1, \tilde{\kappa}^1 : (G \upharpoonright \mathtt{1})@\mathtt{1} \circ .... \circ \Delta_n, \tilde{\kappa}^n : (G \upharpoonright \mathtt{n})@\mathtt{n}$ By rule [Conc]

$\Delta_1, \tilde{\kappa}^1 : (G \upharpoonright \mathtt{1})@\mathtt{1} \circ .... \circ \Delta_n, \tilde{\kappa}^n : (G \upharpoonright \mathtt{n})@\mathtt{n} = \Delta_1 \circ ... \circ \Delta_n, \tilde{\kappa}^1 : (G \upharpoonright \mathtt{1})@\mathtt{1} \circ .... \circ \tilde{\kappa}^n : (G \upharpoonright \mathtt{n})@\mathtt{n}$
$\Gamma \vdash (\nu\tilde{\kappa})(P_1[\tilde{\kappa}^1/\tilde{x}] \mid P_2[\tilde{\kappa}^2/\tilde{x}] \mid ... \mid P_\mathtt{n}[\tilde{\kappa}^n/\tilde{x}]) \triangleright \Delta$ By rule [CRes]

Case: [R-Multicast]

$$\tilde{\kappa}^\mathtt{p}!\langle\tilde{e}\rangle; P \mid \kappa^{\mathtt{p}_1}_{\mathrm{m}_1}?\langle\tilde{x}\rangle; P_1 \mid \cdots \mid \kappa^{\mathtt{p}_\mathrm{n}}_{\mathrm{m}_\mathrm{n}}?\langle\tilde{x}\rangle; P_\mathrm{n} \longrightarrow P \mid P_1\{\tilde{v}/\tilde{x}\} \mid \cdots \mid P_\mathrm{n}\{\tilde{v}/\tilde{x}\}$$
$$(\{\kappa^{\mathrm{m}_1}, ..., \kappa^{\mathrm{m}_\mathrm{n}}\} = \{\tilde{\kappa}\}, \tilde{e} \downarrow \tilde{v})$$

$\Gamma \vdash \tilde{\kappa}^\mathtt{p}!\langle\tilde{e}\rangle; P \mid \kappa^{\mathtt{p}_1}_{\mathrm{m}_1}?\langle\tilde{x}\rangle; P_1 \mid \cdots \mid \kappa^{\mathtt{p}_\mathrm{n}}_{\mathrm{m}_\mathrm{n}}?\langle\tilde{x}\rangle; P_\mathrm{n} \triangleright \Delta$ By assumption
$\Gamma \vdash \tilde{\kappa}^\mathtt{p}!\langle\tilde{e}\rangle; P \triangleright \Delta_1$
$\Gamma \vdash \kappa^{\mathtt{p}_1}_{\mathrm{m}_1}?\langle\tilde{x}\rangle; P_1 \triangleright \Delta_2$
...
$\Gamma \vdash \kappa^{\mathtt{p}_\mathrm{n}}_{\mathrm{m}_\mathrm{n}}?\langle\tilde{x}\rangle; P_\mathrm{n} \triangleright \Delta_{n+1}$
where $\Delta = \Delta_1 \circ ... \circ \Delta_{n+1}$ and $\Delta_1 \asymp ... \asymp \Delta_{n+1}$ By inversion on [Conc]

$\Delta_1 = \Delta'_1, \tilde{\kappa}^\mathtt{p} : \tilde{\mathrm{m}}!\langle\tilde{V}\rangle; T@\mathtt{p}$ By rule [Send]
$\Delta_2 = \Delta'_2, \tilde{\kappa}^{\mathtt{p}_1} : \mathrm{m}_1?\langle\tilde{V}\rangle; T_1@\mathtt{p}_1$ By rule [Rcv]
...
$\Delta_{n+1} = \Delta'_{n+1}, \tilde{\kappa}^{\mathtt{p}_\mathrm{n}} : \mathrm{m}_n?\langle\tilde{V}\rangle; T_n@\mathtt{p}_\mathrm{n}$ By rule [Rcv]

$\Gamma \vdash \tilde{e} \triangleright \tilde{V} \quad \Gamma \vdash P \triangleright \Delta'_1, \tilde{\kappa}^\mathtt{p} : T@\mathtt{p}$ By inversion on [Send]
$\Gamma, \tilde{y} : \tilde{S} \vdash P_1 \triangleright \Delta'_2, \tilde{\kappa}^{\mathtt{p}_1} : T_1@\mathtt{p}_1$ By inversion on [Rcv]
...
$\Gamma, \tilde{y} : \tilde{S} \vdash P_n \triangleright \Delta'_{n+1}, \tilde{\kappa}^{\mathtt{p}_\mathrm{n}} : T_n@\mathtt{p}_\mathrm{n}$ By inversion on [Rcv]

$\Gamma \vdash P_1[\tilde{v}/\tilde{y}] \triangleright \Delta'_2, \tilde{\kappa}^{\mathtt{p}_1} : T_1@\mathtt{p}_1$ By Lemma 2.6.23.1
...

$\Gamma \vdash P_n[\tilde{v}/\tilde{y}] \triangleright \Delta'_{n+1}, \tilde{\kappa}^{\mathtt{p}_n} : T_n @ \mathtt{p}_n$ By Lemma 2.6.23.1

$\Gamma \vdash P \mid P_1[\tilde{v}/\tilde{y}] \mid \cdots \mid P_n[\tilde{v}/\tilde{y}]$
$\triangleright \Delta'_1, \tilde{\kappa}^{\mathtt{p}} : T @ \mathtt{p} \circ \Delta'_2, \tilde{\kappa}^{\mathtt{p}_1} : T_1 @ \mathtt{p}_1 \circ ... \circ \Delta'_{n+1}, \tilde{\kappa}^{\mathtt{p}_n} : T_n @ \mathtt{p}_n$ By rule [Conc]

$\Delta'_1, \tilde{\kappa}^{\mathtt{p}} : T @ \mathtt{p} \circ \Delta'_2, \tilde{\kappa}^{\mathtt{p}_1} : T_1 @ \mathtt{p}_1 \circ ... \circ \Delta'_{n+1}, \tilde{\kappa}^{\mathtt{p}_n} : T_n @ \mathtt{p}_n =$
$\Delta'_1 \circ ... \circ \Delta'_{n+1}, \tilde{\kappa}^{\mathtt{p}} : T @ \mathtt{p} \circ ... \circ \tilde{\kappa}^{\mathtt{p}_n} : T_n @ \mathtt{p}_n$
$\Delta'_1 \circ ... \circ \Delta'_{n+1}, \tilde{\kappa}^{\mathtt{p}} : \mathrm{m}_1, ..., \mathrm{m}_n ! \langle \tilde{V} \rangle . T @ \mathtt{p} \circ \tilde{\kappa}^{\mathtt{p}_1} : \mathrm{m}_1 ? \langle \tilde{V} \rangle . T_1 @ \mathtt{p}_1 \circ ... \circ \tilde{\kappa}^{\mathtt{p}_n} : \mathrm{m}_n ? \langle \tilde{V} \rangle . T_n @ \mathtt{p}_n$
$\to \Delta'_1 \circ ... \circ \Delta'_{n+1}, \tilde{\kappa}^{\mathtt{p}} : T @ \mathtt{p} \circ ... \circ \tilde{\kappa}^{\mathtt{p}_n} : T_n @ \mathtt{p}_n$ By [TR-Context, TR-Mult]

Case: [R-Multicast]

$$\tilde{\kappa}^{\mathtt{p}} ! \langle \tilde{t} \rangle ; P \mid \kappa^{\mathtt{p}_1}_{\mathrm{m}_1} ? \langle \tilde{x} \rangle ; P_1 \mid \cdots \mid \kappa^{\mathtt{p}_n}_{\mathrm{m}_n} ? \langle \tilde{x} \rangle ; P_n \longrightarrow P \mid P_1 \{ \tilde{t}/\tilde{x} \} \mid \cdots \mid P_n \{ \tilde{t}/\tilde{x} \}$$

$$(\{ \kappa^{\mathrm{m}_1}, ..., \kappa^{\mathrm{m}_n} \} = \{ \tilde{\kappa} \})$$

$\Gamma \vdash \tilde{\kappa}^{\mathtt{p}} ! \langle \tilde{t} \rangle ; P \mid \kappa^{\mathtt{p}_1}_{\mathrm{m}_1} ? \langle \tilde{x} \rangle ; P_1 \mid \cdots \mid \kappa^{\mathtt{p}_n}_{\mathrm{m}_n} ? \langle \tilde{x} \rangle ; P_n \triangleright \Delta$ By assumption
$\Gamma \vdash \tilde{\kappa}^{\mathtt{p}} ! \langle \tilde{t} \rangle ; P \triangleright \Delta_1$
$\Gamma \vdash \kappa^{\mathtt{p}_1}_{\mathrm{m}_1} ? \langle \tilde{x} \rangle ; P_1 \triangleright \Delta_2$
$\cdots$
$\Gamma \vdash \kappa^{\mathtt{p}_n}_{\mathrm{m}_n} ? \langle \tilde{x} \rangle ; P_n \triangleright \Delta_{n+1}$
where $\Delta = \Delta_1 \circ \Delta_2 \cdots \circ \Delta_{n+1}$ and $\Delta_1 \asymp . \Delta_2 \cdots \asymp \Delta_{n+1}$ By inversion on [Conc]

$\Delta_1 = \Delta'_1, \tilde{\kappa}^{\mathtt{p}} : \tilde{\mathrm{m}} ! \langle \widetilde{T'} @ p' \rangle ; T @ \mathtt{p},\ \tilde{t} : seq(\widetilde{T'}) @ \mathtt{p}'$ By rule [Thr]
$\Delta_2 = \Delta'_2, \tilde{\kappa}^{\mathtt{p}_1} : \mathrm{m} ? \langle T'_1 @ p' \rangle ; T_1 @ p_1$ By rule [Cat]
$\cdots$
$\Delta_{n+1} = \Delta'_{n+1}, \tilde{\kappa}^{\mathtt{p}_n} : \mathrm{m} ? \langle T'_n @ p' \rangle ; T_n @ p_n$ By rule [Cat]

$\Gamma \vdash P \triangleright \Delta'_1, \tilde{\kappa}^{\mathtt{p}} : T @ \mathtt{p}$ By inversion on [Thr]
$\Gamma \vdash P_1 \triangleright \Delta'_2, \tilde{\kappa}^{\mathtt{p}_1} : T_1 @ \mathtt{p}_1, \tilde{y} : T'_1 @ \mathtt{p}'$ By inversion on [Cat]
$\cdots$
$\Gamma \vdash P_n \triangleright \Delta'_{n+1}, \tilde{\kappa}^{\mathtt{p}_n} : T_n @ \mathtt{p}_n, \tilde{y} : T'_n @ \mathtt{p}'$ By inversion on [Cat]
$\Gamma \vdash P_1[\tilde{t}/\tilde{y}] \triangleright \Delta'_2, \tilde{\kappa}^{\mathtt{p}_1} : T_1 @ \mathtt{p}_1, \tilde{t} : T'_1 @ \mathtt{p}'$ By Lemma 2.6.24
$\cdots$
$\Gamma \vdash P_n[\tilde{t}/\tilde{y}] \triangleright \Delta'_{n+1}, \tilde{\kappa}^{\mathtt{p}_n} : T_n @ \mathtt{p}_n, \tilde{t} : T'_n @ \mathtt{p}'$ By Lemma 2.6.24

$\Gamma \vdash P \mid P_1[\tilde{t}/\tilde{y}] \mid \cdots \mid P_n[\tilde{t}/\tilde{y}] \triangleright$
$\Delta'_1, \tilde{\kappa}^{\mathtt{p}} : T@\mathtt{p} \circ \Delta'_2, \tilde{\kappa}^{\mathtt{p}_1} : T_1@\mathtt{p}_1 \circ \cdots \circ \Delta'_{n+1}, \tilde{\kappa}^{\mathtt{p}_n} : T_n@\mathtt{p}_n, \tilde{t} : seq(\tilde{T}'@\mathtt{p}')$ By rule [Conc]

$\Delta'_1, \tilde{\kappa}^{\mathtt{p}} : T@\mathtt{p} \circ \Delta'_2, \tilde{\kappa} : T_1@\mathtt{p}_1 \cdots \circ \Delta'_{n+1}, \tilde{\kappa} : T_n@\mathtt{p}_n, \tilde{t} : seq(\tilde{T}'@\mathtt{p}') =$
$\Delta'_1 \circ \Delta'_2 \cdots \circ \Delta'_{n+1}, \tilde{t} : seq(\tilde{T}'@\mathtt{p}'), \tilde{\kappa}^{\mathtt{p}} : T@\mathtt{p} \circ \tilde{\kappa}^{\mathtt{p}_1} : T_1@\mathtt{p}_1 \cdots \circ \tilde{\kappa}^{\mathtt{p}_n} : T_n@\mathtt{p}_n$

$\Delta'_1 \circ \Delta'_2 \cdots \circ \Delta'_{n+1}, \tilde{t} : seq(\tilde{T}'@\mathtt{p}'), \tilde{\kappa}^{\mathtt{p}} : \tilde{\mathtt{m}}!\langle \widetilde{T'}@p' \rangle; T@\mathtt{p}, \ \circ \tilde{\kappa}^{\mathtt{p}_1} : \mathrm{m}?\langle T'_1@p' \rangle; T_1@p_1$
$\circ \mathrm{m}?\langle T'_n@p' \rangle; T_n@p_n \ \rightarrow$
$\Delta'_1 \circ \Delta'_2 \cdots \circ \Delta'_{n+1}, \tilde{t} : seq(\tilde{T}'@\mathtt{p}'), \tilde{\kappa}^{\mathtt{p}} : T@\mathtt{p} \circ \tilde{\kappa}^{\mathtt{p}_1} : T_1@\mathtt{p}_1 \cdots$
$\circ \tilde{\kappa}^{\mathtt{p}_n} : T_n@\mathtt{p}_n$ By [TR-Context, TR-MultD]

Case: [R-Label]

$$\tilde{\kappa}^{\mathtt{p}} \oplus \langle l_i \rangle; P \mid \kappa^{\mathtt{p}_1}_{\mathrm{m}_1} \& \{l_j : P_{1j}\}_{j \in I} \mid \cdots \mid \kappa^{\mathtt{p}_n}_{\mathrm{m}_n} \& \{l_j : P_{\mathrm{n}j}\}_{j \in I} \longrightarrow P \mid P_{1i} \mid \cdots \mid P_{\mathrm{n}i}$$

$$(\{\kappa_{\mathrm{m}_1}, ..., \kappa_{\mathrm{m}_n}\} = \{\tilde{\kappa}\}, i \in I)$$

$\Gamma \vdash \tilde{\kappa}^{\mathtt{p}} \oplus \langle l_i \rangle; P \mid \kappa^{\mathtt{p}_1}_{\mathrm{m}_1} \& \{l_j : P_{1j}\}_{j \in I} \mid \cdots \mid \kappa^{\mathtt{p}_n}_{\mathrm{m}_n} \& \{l_j : P_{\mathrm{n}j}\}_{j \in I} \triangleright \Delta$ By assumption
$\Gamma \vdash \tilde{\kappa}^{\mathtt{p}} \oplus \langle l_i \rangle; P \triangleright \Delta_1$
$\Gamma \vdash \kappa^{\mathtt{p}_1}_{\mathrm{m}_1} \& \{l_j : P_{1j}\}_{j \in I} \triangleright \Delta_2$
...
$\Gamma \vdash \kappa^{\mathtt{p}_n}_{\mathrm{m}_n} \& \{l_j : P_{\mathrm{n}j}\}_{j \in I} \triangleright \Delta_{n+1}$ where $\Delta = \Delta_1 \circ ... \circ \Delta_{n+1}$
and $\Delta_1 \asymp ... \asymp \Delta_{n+1}$ By inversion on [Conc]

$\Delta_1 = \Delta'_1, \tilde{\kappa}^{\mathtt{p}} : \tilde{\mathrm{m}} \oplus \{l_j : T_j\}_{j \in I}@\mathtt{p}$ By rule [Sel]
$\Delta_2 = \Delta'_2, \tilde{\kappa}^{\mathtt{p}_1} : \mathrm{m}_1 \& \{l_j : T_{1j}\}_{j \in I}@\mathtt{p}_1$ By rule [Br]
...
$\Delta_{n+1} = \Delta'_{n+1}, \tilde{\kappa}^{\mathtt{p}_n} : \mathrm{m}_n \& \{l_j : T_{nj}\}_{j \in I}@\mathtt{p}_n$ By rule [Br]

$\Gamma \vdash P \triangleright \Delta'_1, \tilde{\kappa}^{\mathtt{p}} : T_i@\mathtt{p}$ and $i \in I$ By inversion on [Sel]
$\forall j \in I, \Gamma \vdash P_{1_j} \triangleright \Delta'_2, \tilde{\kappa}^{\mathtt{p}_1} : T_{1_j}@\mathtt{p}_1$ By inversion on [Br]
...
$\forall j \in I, \Gamma \vdash P_{n_j} \triangleright \Delta'_{n+1}, \tilde{\kappa}^{\mathtt{p}_n} : T_{n_j}@\mathtt{p}_n$ By inversion on [Br]

$\Gamma \vdash P \mid P_{1i} \mid \cdots \mid P_{ni} \triangleright \Delta'_1 \circ ... \circ \Delta'_{n+1}, \tilde{\kappa}^{\mathtt{p}_1} : T_i@\mathtt{p} \circ ... \circ \tilde{\kappa}^{\mathtt{p}_n} : T_{n_i}@\mathtt{p}_n$ By rule [Conc]

$\Delta'_1, \tilde{\kappa}^{\mathtt{p}} : \mathrm{m}_1, ..., \mathrm{m}_n \oplus \{l_j : T_j\}_{j \in I}@\mathtt{p}, \Delta'_2, \tilde{\kappa}^{\mathtt{p}_1} : \mathrm{m}_1 \& \{l_j : T_{1j}\}_{j \in I}@\mathtt{p}_1, ...,$
$\Delta'_{n+1}, \tilde{\kappa}^{\mathtt{p}_n} : \mathrm{m}_n \& \{l_j : T_{nj}\}_{j \in I}@\mathtt{p}_n \ \rightarrow$
$\Delta'_1 \circ ... \circ \Delta'_{n+1}, \tilde{\kappa}^{\mathtt{p}} : T_i@\mathtt{p} \circ ... \circ \tilde{\kappa}_{\mathtt{p}_n} : T_{n_i}@\mathtt{p}_n$ By [TR-Context, TR-MultL]

Case: [R-IfT] and [R-IfF] are trivial by induction.

Case: [R-ProcCall]

$$\mathsf{def}\ D\ \mathsf{in}\ (X\langle \tilde{e} \rangle \mid Q) \ \rightarrow \ \mathsf{def}\ D\ \mathsf{in}\ (P[\tilde{v}/\tilde{y}] \mid Q) \qquad (\tilde{e} \downarrow \tilde{v}, X(\tilde{y}) = P \in D)$$

$\Gamma \vdash \mathsf{def}\ D\ \mathsf{in}\ (X\langle \tilde{e} \rangle \mid Q) \triangleright \Delta$ By assumption
$\Gamma, X : \tilde{S}\tilde{T}, \tilde{y_1} : \tilde{S} \vdash P \triangleright \tilde{y_2} : \tilde{T}@\tilde{\mathtt{p}}$, $\Gamma, X : \tilde{S}\tilde{T} \vdash X\langle \tilde{e} \rangle \mid Q \triangleright \Delta$
where $\tilde{y_1} \in \tilde{y}$ and $\tilde{y_2} \in \tilde{y}$ By inversion on rule [Def]
$\Gamma, X : \tilde{S}\tilde{T} \vdash X\langle \tilde{e} \rangle \triangleright \Delta_1$ and $\Gamma, X : \tilde{S}\tilde{T} \vdash Q \triangleright \Delta_2$
where $\Delta = \Delta_1 \circ \Delta_2$ and $\Delta_1 \asymp \Delta_2$ By inversion on rule [Conc]
$\Gamma \vdash \tilde{e_1} \triangleright \tilde{S}$ and $\Delta_1 = \Delta'_1, \tilde{\kappa} : \tilde{T}@\tilde{\mathtt{p}}$ where $\tilde{\kappa}, \tilde{e_1} \in \tilde{e}$ By inversion on rule [Var]
$\Gamma, X : \tilde{S}\tilde{T} \vdash P[\tilde{v}/\tilde{y}] \triangleright \tilde{\kappa} : \tilde{T}@\tilde{\mathtt{p}}$ By Lemma 2.6.23.1 and Lemma 2.6.24
$\Gamma, X : \tilde{S}\tilde{T} \vdash P[\tilde{v}/\tilde{y}] \triangleright \Delta'_1, \tilde{\kappa} : \tilde{T}@\tilde{\mathtt{p}}$ By Lemma 2.6.23.2
$\Gamma, X : \tilde{S}\tilde{T} \vdash P[\tilde{v}/\tilde{y}] \mid Q \triangleright \Delta'_1, \tilde{\kappa} : \tilde{T}@\tilde{\mathtt{p}} \circ \Delta_2 = \Delta$ By rule [Conc]
$\Gamma \vdash \mathsf{def}\ D\ \mathsf{in}\ (P[\tilde{v}/\tilde{y}] \mid Q) \triangleright \Delta$ By rule [Def]

Case: [R-Par]

$$P \ \rightarrow \ P' \ \Rightarrow \ P \mid Q \ \rightarrow \ P' \mid Q$$

$\Gamma \vdash P \mid Q \triangleright \Delta$ By assumption
$\Gamma \vdash P \triangleright \Delta_1$ and $\Gamma \vdash Q \triangleright \Delta_2$ where $\Delta = \Delta_1 \circ \Delta_2$ and $\Delta_1 \asymp \Delta_2$ By rule [Conc]
$\Gamma \vdash P' \triangleright \Delta'_1$ where $\Delta_1 = \Delta'_1$ or $\Delta_1 \ \rightarrow \ \Delta'_1$ By induction
when $\Delta_1 = \Delta'_1$ then the proof is trivial so we investigate the second case
when $\Delta_1 \ \rightarrow \ \Delta'_1$
$\Delta_2 \asymp \Delta'_1$ and $\Delta_1 \circ \Delta_2 \ \rightarrow \ \Delta'_1 \circ \Delta_2$ By Lemma 2.6.36 (1)
$\Gamma \vdash P' \mid Q \triangleright \Delta'_1 \circ \Delta_2$ By rule [Conc]

Case: [R-Scop, R-CProcCall] is trivial by induction.

Case: [R-StructC]

$$P \equiv P' \text{ and } P' \ \rightarrow \ Q' \text{ and } Q' \equiv Q \ \Rightarrow \ P \ \rightarrow \ Q$$

$\Gamma \vdash P \triangleright \Delta$, $P \equiv P'$, $P' \ \rightarrow \ Q'$ and $Q' \equiv Q$	By assumption
$\Gamma \vdash P' \triangleright \Delta$, $P' \ \rightarrow \ Q'$ and $Q' \equiv Q$	By Theorem 2.6.37
$\Gamma \vdash Q' \triangleright \Delta$, $Q' \equiv Q$	By induction
$\Gamma \vdash Q \triangleright \Delta$	By Theorem 2.6.37

□

We say:

- A prefix is *at* k (resp. *at* a) if its subject (i.e. its initial channel) is k (resp. a). Further a prefix is *emitting* if it is request, output, delegation or selection, otherwise it is *receiving*.
- A prefix is *active* if it is not under a prefix or an `if` branch, after any unfoldings by [Def]. We write $P\langle k \rangle$ if P contains an active subject at k after applying [Def], and $P\langle k! \rangle$ (resp. $P\langle k? \rangle$) if P contains an emitting (resp. receiving) active prefix at k.
- P has a *redex at* k if it has an active emitting and receiving prefix at k among its redexes.

Lemma 2.6.39. Assume $\Gamma \vdash P \triangleright \Delta$ s.t. $\Delta \circ \Delta_0$ is coherent for some Δ_0.

1. If $P\langle k \rangle$ then P contains either a unique active prefix at k or a unique active emitting prefix and a unique active receiving prefix at k.
2. If P contains an active emitting (resp. receiving) prefix at k then Δ contains an emitting (resp. receiving) prefix at k.

Proof. We prove the following statement which implies both (1) and (2) by rule induction on the typing rules. Recall Δ is partially coherent when for some Δ_0 we have $\Delta \asymp \Delta_0$ and $\Delta \circ \Delta_0$ is coherent.

Assume $\Gamma \vdash P \triangleright \Delta$ s.t. Δ is partially coherent. Assume $P\langle k \rangle$. Then one of the following conditions holds.

(a) P contains a unique active receiving (resp. emitting) prefix at k and Δ contains a unique receiving (resp. emitting) prefix at k.

(b) P contains a unique receiving prefix at k and a unique emitting prefix at k. Moreover Δ contains a unique receiving prefix at k and a unique emitting prefix at k.

Case [Mcast], [Macc]: Vacuous since in this case the unique active prefix in the process is at a shared name.

Case [Send], [Rcv], [Thr], [Cat], [Sel] and [Br]: Immediate since there can only be a unique active channel name which is by the given prefixing.

Case [Inact], [If], [Var], [Def]: Vacuous.

Case [Conc]: Suppose

$$\Gamma \vdash P \triangleright \Delta, \qquad \Gamma \vdash Q \triangleright \Delta'$$

$\Delta \asymp \Delta'$. Observe if $\Delta \circ \Delta'$ is partially coherent then Δ and Δ' respectively are partially coherent by definition. By induction hypothesis we can assume P and Q satisfy the required condition.

1. If only one party has an active prefix at k there is nothing to prove.
2. If both are active at k, suppose both processes, hence Δ and Δ', have receiving active prefixes at k. Then this cannot be partially coherent since if so then the assumed completion of $\Delta \circ \Delta'$ to a coherent typing should also contain two receiving prefixes which are impossible by Proposition 2.6.35 (2, 3). Similarly when two include active emitting prefixes at s, resulting in a contradiction.

Note that this pair may *not* be a redex: we do not (have to) validate coherence until we hide channels, however it is important that there is one output and one input since if not there will be a conflict at k.

Case [NRes]: Vacuous since there is no change either in the process nor in the typing.

Case [CRes]: Vacuous since there is no difference in the typing for k nor in the activeness in prefixes.

Case <:: Vacuous again. □

Definition 2.6.40. The runtime $\mathcal{E}$ with the hole "[]" defines the evaluation contexts of processes as:

$\mathcal{E}$ ::=		Processes evaluation contexts
	$[_]$	Hole
|	$\mathcal{E} \mid P$	Left Parallel
|	$P \mid \mathcal{E}$	Right Parallel
|	$(\boldsymbol{\nu} n)\mathcal{E}$	Hiding
|	$\mathsf{def}\ D\ \mathsf{in}\ \mathcal{E}$	Recursion

The definition and proof of communication-safety below is in the style of Yoshida and Vasconcelos (2007); Vasconcelos et al. (2004).

Definition 2.6.41. Process P is *non-linear* if $P \equiv \mathsf{def}\ D\ \mathsf{in}\ (\boldsymbol{\nu}\tilde{w})(Q)$ where Q is, for some k_{m}, the parallel composition of three or more processes having active prefixes, as $Q_i\langle k_{\mathrm{m}}\rangle, i \in [1..n], n \geq 3$ s.t. $Q \equiv Q_1 \mid Q_2 \mid Q_3 \mid ... \mid Q_n$.

Definition 2.6.42. Process P is *error* if $P \equiv \mathsf{def}\ D\ \mathsf{in}\ (\boldsymbol{\nu}\tilde{w})(Q)$ where Q is the parallel composition of two or more processes having active prefixes, as $Q_i\langle k_i\rangle, i \in [1..n], n \geq 2$, where $Q \equiv Q_1 \mid Q_2 \mid ... \mid Q_n$ that do not form a redex.

Theorem 2.6.43 (Communication-safety)**.** A typeable program with a coherent Δ never reduces to a non-linear and error process.

Proof. By type preservation, it suffices to show that typable programs are not non-linear and errors. The proof is by reductio ad absurdum, assuming non-linear and error processes are typable with a coherent Δ. Suppose that $\Gamma \vdash \mathsf{def}\ D\ \mathsf{in}\ (\boldsymbol{\nu}\tilde{w})(P) \triangleright \Delta$ with Δ coherent. Analysing the derivation tree for the process, we conclude that $\Gamma \vdash P \triangleright \Delta$. We analyse first non-linearity and then the second class of error.

When P is the parallel composition of three or more processes having active prefixes $P_i\langle k_{\mathrm{m}}\rangle, i \in [1..n], n \geq 3$ s.t. $P \equiv P_1 \mid P_2 \mid P_3 \mid ... \mid P_n$, we concentrate on the case of three processes; the remaining cases are the same. Let $P \equiv P_1 \mid P_2 \mid P_3$ then at least two processes have respectively an active emitting or receiving prefixes on k_{m}. If no two processes $(P_1 \mid P_2, P_2 \mid P_3, P_1 \mid P_3)$ form a redex on k_{m} then we have a contradiction by the second part of the theorem (Note that the three processes can not form a redex in multicast as a prefix such as $k_{\mathrm{m}}, k_{\mathrm{m}}!/\oplus$ is not defined by the syntax of processes). Otherwise we have two active prefixes forming a redex. Then, we have the following Δ by Lemma 2.6.39(2), we have a case of two active emitting and a case of two acctive receiving

prefixes on k_{m} in $\Delta = \Delta_1 \circ \Delta_2 \circ \Delta_3$: $\mathrm{m}!\langle U\rangle; T@\mathtt{p}, \mathrm{m}?\langle U\rangle; T'@\mathtt{p}', \mathrm{m}!\langle U\rangle; T''@\mathtt{p}'' \in \Delta(\tilde{k})$ or $\mathrm{m}!\langle U\rangle; T@\mathtt{p}, \mathrm{m}?\langle U\rangle; T'@\mathtt{p}', \mathrm{m}?\langle U\rangle; T''@\mathtt{p}'' \in \Delta(\tilde{k})$ (The branching case is symmetric). By assumption, we have that $\Delta(\tilde{k}) = [\![G]\!]$ is coherent; thus, G is coherent. By Proposition 2.6.35(3) we have that the casual edges in G are preserved in $[\![G]\!] = \Delta(\tilde{k})$. Since no output chain for the first case and input chain for the second case can be defined between the prefixes of $\Delta(k)$, then G is not linear on k_{m}. Hence, G is not coherent. Thus, we have a contradiction that $\mathsf{def}\ D\ \mathsf{in}\ (\boldsymbol{\nu}\tilde{w})(P)$ is typeable with a coherent Δ.

For the other cases of more than three processes, the structure of the proof is the same. Firstly, we define the two cases when no subgroup of two processes forms a redex and two processes form a redex. For the first case, we have immediately a contradiction by the second part of the theorem. Secondly, if two processes form a redex then there will be another process with an active emitting or receiving prefix that makes G non-linear, resulting in a contradiction that $\mathsf{def}\ D\ \mathsf{in}\ (\boldsymbol{\nu}\tilde{w})(P)$ is typeable with a coherent Δ.

When P is the parallel composition of two or more processes having active prefixes $P_i\langle\!\langle k_i \rangle\!\rangle, i \in [1..n], n \geq 2$ s.t. $P \equiv P_1 \mid P_2 \mid ... \mid P_n$ that do not form a redex. We concentrate on the case of two processes; the remaining cases are the same. Let $P \equiv P_1 \mid P_2$ then the cases are the following:

Case: Send-Select By rule [Conc], we have that $\Gamma \vdash P_1 \rhd \Delta_1$ and $\Gamma \vdash P_2 \rhd \Delta_2$ where $\Delta = \Delta_1 \circ \Delta_2$. By rule [Send] and [Sel], we have that $\mathrm{m}!\langle U\rangle; T@\mathtt{p} \in \Delta_1(\tilde{k})$ and $\mathrm{m}' \oplus \{l_i : T'_i\}_{i\in I}@\mathtt{p}' \in \Delta_2(\tilde{k})$; thus, $\tilde{k} : \mathrm{m}!\langle U\rangle; T@\mathtt{p}, \mathrm{m}' \oplus \{l_i : T'_i\}_{i\in I}@\mathtt{p}' \in \Delta$ (The case of typing the sending construct with the [Thr] rule is symmetric). By assumption, we have that $\Delta(\tilde{k}) = [\![G]\!]$ is coherent; thus, G is coherent. The two prefixes m! and m′⊕ can not form a causality in the syntax of global types. Thus, the other case defines the prefixes (with casual edges in each end-point type as $(\mathsf{OI}, \mathsf{OI})$) in two different causalities that cannot be ordered. This is impossible by the definition of global types, resulting in a non-coherent Δ. Hence, we have a contradiction that $\mathsf{def}\ D\ \mathsf{in}\ (\boldsymbol{\nu}\tilde{w})(P)$ is typeable with a coherent Δ.

The other cases [Send-Send], [Selection-Selection], [Send-Branch] are symmetric. The other case [Reception-Select] is symmetric with the difference that the prefixes have casual edges IO, OI in each end-point type. The other cases [Reception-Reception], [Reception-Branch] are symmetric with the difference that the prefixes have casual edges IO, IO.

Case: Send-Receive By rule [Conc], we have that $\Gamma \vdash P_1 \rhd \Delta_1$ and $\Gamma \vdash P_2 \rhd \Delta_2$ where $\Delta = \Delta_1 \circ \Delta_2$. By rule [Send] and [Sel], we have that $\mathrm{m}!\langle U\rangle; T@\mathtt{p} \in \Delta_1(\tilde{k})$ and

$\mathrm{m}'?\langle U'\rangle@\mathtt{p}' \in \Delta_2(\tilde{k})$; thus, $\tilde{k} : \mathrm{m}!\langle U\rangle; T@\mathtt{p}, \mathrm{m}'?\langle U'\rangle@\mathtt{p}' \in \Delta$ (The case of typing the sending construct with the [Thr], [Br] rule is symmetric). The two prefixes m! and m′? can not form a causality in the syntax of global types as $\mathrm{m} \neq \mathrm{m}'$ and $U \neq U'$. For the other case that the prefixes are part of two co-occurring prefixes in G, then the reasoning is the same as in the above case with the difference that prefixes have casual edges OO, II.

Case: MSend-Receive-Receive By rule [Conc], we have that $\Gamma \vdash P' \triangleright \Delta', \Gamma \vdash P_i \triangleright \Delta_i$ where $P \equiv P' \mid P_1 \mid P_2$ and $\Delta = \Delta' \circ \Delta_1 \circ \Delta_2$. By rule [Send] and [Rcv], we have that $\mathrm{m}_1, ..., \mathrm{m}_{n'}!\langle U\rangle; T@\mathtt{p} \in \Delta'(\tilde{k})$ and $\mathrm{m}_i?\langle U\rangle; T_i@\mathtt{p}_i \in \Delta_i(\tilde{k}), i = [1, 2]$, where $n' > 2$; thus, $\tilde{k} : \mathrm{m}_1, ..., \mathrm{m}_{n'}!\langle U\rangle; T@\mathtt{p}, \mathrm{m}_i?\langle U\rangle; T_i@\mathtt{p}_i \in \Delta, i = [1, 2]$ (The case of typing the sending construct with the [Thr], [Catch] rule is symmetric). The two prefixes $\mathrm{m}_1, ..., \mathrm{m}_{n'}!$ and $\mathrm{m}_1?, \mathrm{m}_2?$ can not form a causality in the syntax of global types as $n' > 2$. For the other case that the prefixes are part of co-occurring prefixes in G, then the reasoning is the same as in the above case with the difference that prefixes have casual edges $\mathsf{OO}, ...\mathsf{OO}, \mathsf{II}, \mathsf{II}$. When there are more than two processes, say n, having an active receiving prefix such that $n' > n$ then this case applies the same.

Case [MSel-Branch-Branch] is symmetric.

Case: Select-Branch By rule [Conc], we have that $\Gamma \vdash P_1 \triangleright \Delta_1$ and $\Gamma \vdash P_2 \triangleright \Delta_2$ where $\Delta = \Delta_1 \circ \Delta_2$. By rule [Sel] and [Br], we have that $\mathrm{m} \oplus \{l_i : T_i\}_{i\in I}@\mathtt{p} \in \Delta_1(\tilde{k})$ and $\mathrm{m}'\&\{l'_i : T'_i\}_{i\in J}@\mathtt{p}' \in \Delta_2(\tilde{k})$; thus, $\tilde{k} : \mathrm{m} \oplus \{l_i : T_i\}_{i\in I}@\mathtt{p}, \mathrm{m}'\&\{l'_i : T'_i\}_{i\in J}@\mathtt{p}' \in \Delta$. The two prefixes m⊕ and m′& cannot form a causality in the syntax of global types as $\mathrm{m} \neq \mathrm{m}'$ and $l_i \neq l'_i, I \neq J$. For the other case that the prefixes are part of two co-occurring prefixes in G, then the reasoning is the same as in the above case with the difference that prefixes have casual edges OO, II.

For the other cases of more than two processes, the structure of the proof is the same with the difference that the sending, reception, selection, branching, constructs are repeated more than once in each of the forementioned cases. Firstly, we define the two cases when the i prefixes in $\Delta(\tilde{k})$ can or cannot form a causality in the syntax of global types. For the first case, we need to observe only two (or more in case of multicast) prefixes as in the cases discussed above. For the other case, we will observe that the prefixes (for the [Send-Select] case, the casual edges in each end-point type are OI in co-occurring causalities cannot be ordered, resulting in a non-coherent Δ. Hence, we have a contradiction that $\mathsf{def}\ D\ \mathsf{in}\ (\boldsymbol{\nu}\tilde{w})(P)$ is typeable with a coherent Δ. □

Proposition 2.6.44. Let Δ be coherent and $\Delta(\tilde{k}) = [\![G]\!]$. Then $\Delta \xrightarrow{\tilde{\mathrm{m}}} \Delta'$ iff $G \xrightarrow{\tilde{\mathrm{m}}} G'$ with

$\Delta'(\tilde{k}) = [\![G']\!]$ where $k_{\tilde{\mathrm{m}}} \in \tilde{k}$.

Proof. The $\Rightarrow$ implication, follows directly from the definition 2.6.28 of global types reduction and that $\Delta \xrightarrow{\tilde{\mathrm{m}}} \Delta'$ implies $\Delta(\tilde{k}) \xrightarrow{\tilde{\mathrm{m}}} \Delta'(\tilde{k})$. For the $\Leftarrow$ implication, by Proposition 2.6.35(3) the casual edges of G are preserved in the images of $[\![G]\!]$ and by Corollary 2.6.33, there is a bijection between prefixes in G and the set of input-output prefixes in $[\![G]\!]$. From the proposition and corollary, it follows that $[\![G]\!] \xrightarrow{\tilde{\mathrm{m}}} [\![G']\!]$ where $\Delta(\tilde{k}) = [\![G]\!]$ and $\Delta'(\tilde{k}) = [\![G']\!]$. By rule [TR-Context] follows the conclusion of the implication. □

Corollary 2.6.45 (session fidelity)**.** Assume $\Gamma \vdash P \triangleright \Delta$ such that Δ is coherent and $\Delta(\tilde{k}) = [\![G]\!]$. If $P\langle\!\langle k_{\tilde{\mathrm{m}}} \rangle\!\rangle \to P'$ at the redex of $k_{\tilde{\mathrm{m}}}$, then $\Gamma \vdash P' \triangleright \Delta'$ with $G \xrightarrow{\tilde{\mathrm{m}}} G'$ and $[\![G']\!] = \Delta'(\tilde{k})$.

Proof. The former conclusion $\Gamma \vdash P' \triangleright \Delta'$ where $\Delta = \Delta'$ or $\Delta \xrightarrow{\tilde{\mathrm{m}}} \Delta'$ follows directly from the subject reduction theorem 2.6.38. The late conclusions $G \xrightarrow{\tilde{\mathrm{m}}} G'$ and $[\![G']\!] = \Delta'(\tilde{k})$ follow directly from Proposition 2.6.44. □

Below, we provide as a conjecture the property of progress: a process has the progress property if in all configurations where it is provided a suitable context either (1) it does not contain session channels or (2) it can be further reduced.

Conjecture 2.6.46 (Progress)**.** A process P has the progress property if $P \to^\star P'$ implies either P' does not have session channel or $P'|P'' \to$ for some P'' such that $P'|P''$ is well-typed and $P'' \not\to$.

Although, we do not have a formal proof, we believe the system satisfies the standard progress property. Indeed, our system benefits from the proof of progress for well-typed processes (session initiation, conditional, parallel composition, inaction, hiding and recursion) of asynchronous communications by Honda et al. (2008b). To complete the proof, we need to ascertain that a well-typed program of exchaning primitive values, labels, and channels reduces further. We leave the proof of progress for future work.

2.7 Summary

CSMS extends the synchronous π-calculus with constructs that model structured interactions between a fixed number of processes, in a similar way to the previous work by

Honda *et al.* which extended the asynchronous π-calculus. Thus, CSMS completes the work on multiparty session types for the remaining mathematical dialect of the π-calculus. In this chapter, we have observed the nature of synchronous communications in modeling control for timing events and strong sequential order of messages. In addition, CSMS overcomes some of the shortfalls of the previous work by introducing a simpler syntax of processes, a concept of multicast send of values and labels, a practical model of higher-order communications, and a more simple and declarative version of linearity.

In the previous system, a different primitive was used to send channels to support higher-order communication, mismatching the constructs of actual programming languages such as Java and SJ where a single primitive is used to send data of any type. However in CSMS, only a single sending primitive is used to deliver data. In the previous system, higher-order communication is modeled with the receiver of the channels possessing them before the communication has taken place, mismatching the constructs of actual languages that support communication and so, making the theory inapplicable to them. However in CSMS, the receiver possesses the channels only after the communication takes place. This is safely provided, i.e. holding preservation, through multipolarity labels—syntax that denotes the process which the channel belongs to. In the previous systems, there is a lack of primitives for simultaneous delivery of data, resulting in less productive code and less expressive global types when it comes to branching. In contrast CSMS supports multicast send of values and labels, increasing the productivity and expressivity of global types. In the previous work, the convention of invisibility of delegation in the definition of global types resulted in ill-typed correct programs by the type system. However in this system, invisibility of delegation is a clear convention to follow when defining delegation. A more declarative and minimal linearity property assures a global type whether the use of a same channel in two different communications breaks, at runtime, the causalities specified in the global description. The type system of CSMS typechecks the processes of a session by the types returned from projection of global type onto participants. The CSMS type system is proved to be sound with respect to the operational semantics, including type preservation and communication-safety, and coherent with respect to the global types.

In the next chapter, we introduce parameters to multiparty session types to express naturally and safely richer patterns of communication and programming constructs.

Chapter 3

Parameterised Session Types

This chapter introduces the Calculus of Parameterised Sessions: CPS and its type system. CPS is based on Bettini et al. (2008)'s system to describe sessions involving an arbitrary number of participants, typically represented as communication patterns. The features of Bettini et al. (2008)'s system were discussed in the introduction. We have augmented the syntax of global types with parameters that abstract the number of participants and an iterative construct that builds instances of communication patterns. A programming idiom of role —a concept similar to *class*— is introduced, presenting a study that matches constructs of mainstream languages and so, a way to implement parameterised session types. Our formal system allows programmers to represent parameterised sessions (communication patterns) by global types and then validate roles by type-checking.

Section 3.1 introduces the programming idiom of role and discusses similarity with the idiom of class through the Ring pattern. Section 3.2 shows how this system can represent various communication patterns and control the main source of programming errors in MPI: index calculation. Section 3.3 illustrates the practical utility of our system through real-world examples from parallel algorithms and key distribution protocol. The formal model of CPS, including the syntax of global types and roles, operational semantics, well-formedness, type system and properties of the latter is given in Section 3.4. A comparison of this system and another system from a parallel work is discussed in Section 3.5.

3.1 Roles

A user-defined program in CPS is a global type and a function from natural numbers (the number of participants) to a fixed number of roles composed in parallel, while a run-time program is a fixed number of processes (process in the same meaning as in CSMS) composed in parallel forming a session. Each role of the session defines a behaviour that is unique but shared by many processes at runtime. This section gives an informal introduction of role through the Ring pattern. Next, we provide the definition of role:

Definition 3.1.1 (Role)**.** A *role* defines an abstraction of end-points' communication in mobile processes. It is a blueprint that describes the nature of a communication pattern and the behaviour that all run-time processes share.

A role has the same structure as CSMS processes: it typically starts with a session prefix over a session identifier, followed by a sequence of sending and receiving actions. For example, the Ring pattern as described in the introduction, has three distinct roles: *Starter*, represented by $\mathtt{W}[0]$, *Middle*, represented by $\mathtt{W}[i]$, and *Last*, represented by $\mathtt{W}[\mathrm{n}]$. Below, we provide the main program and roles of the Ring. To ensure the presence of all three roles in a session, we set the number of participants to $n{\geq}2$.

$$
\begin{aligned}
&\mathtt{def}\ \ W = \mathbf{R}\ \mathtt{W}[n]\ \lambda i.\lambda X.\mathtt{W}[i+2], X\ (n-2)\\
&\mathit{Starter} \triangleq \bar{a}[\mathtt{W}[0], \mathtt{W}[1], \mathtt{W}](y).y!\langle \mathtt{W}[1], v\rangle; y?(\mathtt{W}[n], z); R\\
&\mathit{Middle(i)} \triangleq a[\mathtt{W}[i+1]](y).y?(\mathtt{W}[i], z); y!\langle \mathtt{W}[i+2], z\rangle; R'\\
&\mathit{Last} \triangleq a[\mathtt{W}[n]](y).y?(\mathtt{W}[n-1], z); y!\langle \mathtt{W}[0], z\rangle; S\\
&\mathrm{Ring} \triangleq \lambda n.((\mathbf{R}\ \mathit{Starter} \mid \mathit{Last}\ \lambda i.\lambda X.\mathit{Middle(i)} \mid X)\ (n-1))
\end{aligned}
$$

where W denotes the parameterised list of principals $\mathtt{W}[2]$, ..., $\mathtt{W}[n]$ mathematically represented through the $\mathbf{R}$ operator; $k!\langle \mathtt{p}, v\rangle$ denotes the action of sending the value v to participant $\mathtt{p}$ through channel k; $k?\langle \mathtt{p}, x\rangle$ denotes the action of receiving a value place-held by x from participant $\mathtt{p}$ through channel k. Roles *Starter* and *Last* are parameterised by parameter n and *Middle* by index i; both parameter and index are special variables in the calculus. *Middle* is composed in parallel with the process variable X that is used as a placeholder of processes created at each iteration of $\mathbf{R}$ by substituting i with a predecessor of n (number of workers applied to the lambda term); in the last iteration for n=0, X will be replaced with processes of *Starter* and *Last*, obtained by replacing n with the natural number n. The fore-mentioned operations are all provided by the semantics of the $\mathbf{R}$ operator, defined below:

Definition 3.1.2. The semantics of the **R** operator is given via the reduction relation $\longrightarrow$, written $\mathrm{E} \longrightarrow \mathrm{E}'$, read "term E reduces to term E′ in one step".

$$\mathbf{R}\ S\ \lambda i.\lambda X.R\ 0 \longrightarrow S \qquad \text{[Zero]}$$

$$\mathbf{R}\ S\ \lambda i.\lambda X.R\ \mathrm{n}+1 \longrightarrow R\{\mathrm{n}/i\}\{\mathbf{R}\ S\ \lambda i.\lambda X.R\ \mathrm{n}/X\} \qquad \text{[Succ]}$$

where rule [Zero] returns the behaviour S and defines the last iteration of the **R** operator; rule [Succ] replaces each occurrence of the index i in R with a predecessor of n+1 and replaces X with instances of R returned by the other iterations.

The behaviour of the processes created is the same as in CSMS— a private sequence of communications. The shared name a is composed by W. The reduction steps of the Ring for n=2 are given in Section 3.4.2.

In this paragraph, we discuss the similarities between the role and class programming idioms through the *Starter* role. Below, we give the implementation of the latter in Java. The role is naturally implemented in a particular class and so are the other roles, avoiding the MPI style of programming Single Program Multiple Data.

```
public class Starter{
   public Starter(int port_l, String host_r, int port_r){
      // Declare sockets and streams.
      ServerSocket serverSocket = null;
      Socket clientSocket = null;
      PrintWriter out = null;
      BufferedReader in = null;
      try{
          // Set up the sockets for the pattern.
          serverSocket = new ServerSocket(port_l);
          clientSocket = new Socket(host_r, port_r);
          out = ...; // Initialise the output stream on clientSocket.
          in = ...; // Initialise the input stream on serverSocket.
          // Exchange messages with neighbors.
          out.println("1");
          String m = in.readline();
          ... // (1) close streams and sockets, (2) capture exceptions.
   }
}
```

The class `Starter`, in a similar way to the role, provides a communication abstraction of the session object that will be created at run-time given the values for the parameters `port_l`, `host_r`, `port_r`. The parameters abstract the neighbors' address and ports, similarly as n (right neighbor) in the role definition. The classes of the Ring may run on different machines manually. That is, programmers can run the instances of Starter and Last on two distinct machines, and instances of Middle on other different machines; this approach requires that each class defines a main method to create instances of that class. Another alternative is to follow the MPJ Express (Shafi (2006)) approach, where programmers specify the number of instances and hosts where to run each role in a special class, and finally run that class; MPJ Express is a Java messaging system based on the MPI standard. The feature that does not match the Java model with that of our calculus is the communication abstraction. Java uses client-server sockets as communication abstraction, while in our calculus, we use session channels s that define communication between a group of processes, including the n-party handshake. Thus, a Java library that supports communication over a group of processes, provides the proper features that model roles and session channels and so, the proper framework to implement parameterised session types. Possible communication entities over group of processes are discussed in Section 5.2.1.

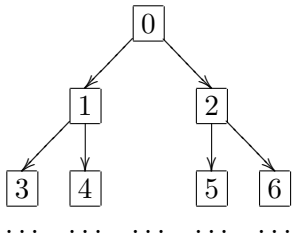
3.2 Communication Patterns

This section illustrates various communication patterns such as Tree, Star and Mesh in CPS. As mentioned in the introduction, communication patterns are important to our study due to their extensive use in the design of many parallel computing architectures of parallel algorithms, data exchange protocols and web-services. The programming methodology of CPS is similar as the one of CSMS: programmers first define the global type for the intended pattern and secondly program each role of the session.

3.2.1 Tree Pattern

The Tree pattern consists of $2^{n+1}-1$ workers organised in a binary tree, numbered in the Ahnentafel system; the diagram in Figure 3.1 is numbered for n =2. The global type next to the diagram specifies a message exchange between a parent and children, where 2^n-1 is the number of internal nodes, including the root, and n denotes the depth. The first message sent is from $\mathtt{W}[2^{\mathrm{n}}-2]$ to $\mathtt{W}[2^{\mathrm{n}+1}-3]$ and the last is sent from $\mathtt{W}[0]$ to $\mathtt{W}[1]$ and so,

the internal nodes, firstly, send to their children and, secondly, receive from their parent.



$\mathbf{R}$ end $\lambda j.\lambda y.\mathtt{W}[j] \to \mathtt{W}[2{*}j{+}1]:\langle U\rangle.$
$\mathtt{W}[j] \to \mathtt{W}[2{*}j{+}2]:\langle U\rangle.y\ (2^n{-}1)$

Figure 3.1: CPS: Diagram and global type of the Tree pattern.

A tree has three kind of nodes: root, internal and leaf. The process running on the root sends a message to its children; the processes on internal nodes send a message to their children and receive a message from their parents as explained above; the processes on leaf nodes receive a message from their parents. The three types of node define three distinct roles of the Tree. An internal or leaf node is enumerated by an even or odd number, and thus the mathematical expressions that identify the parent and children of each of these nodes are different. For this reason, even and odd nodes define two distinct roles in the same kind of node: internal/leaf. Thus, we distinguish five roles in the Tree: *Root* represented by $\mathtt{W}[0]$, *OddInt* and *EvenInt* by $\mathtt{W}[2{*}i{+}1]$ and $\mathtt{W}[2{*}i{+}2]$ $(0{\leq}i{\leq}2^{n-1}{-}2)$, and, *OddLeaf* and *EvenLeaf* by $\mathtt{W}[2{*}i{+}1]$ and $\mathtt{W}[2{*}i{+}2]$ $(2^{n-1}{-}1{\leq}i{\leq}2^n{-}2)$. Below, we provide the Tree roles and the main program. To ensure the presence of all five roles in a session, we set $n \geq 2$.

$\mathtt{def}\ W = \mathbf{R}\ \mathtt{W}[2^{n+1}-2]\ \lambda i.\lambda X.(\mathtt{W}[i+2], X)\ 2^{n+1}-4$

$\mathit{Root} \triangleq \bar{a}[\mathtt{W}[0], \mathtt{W}[1], \mathtt{W}](y).y!\langle \mathtt{W}[1], f(1)\rangle; y!\langle \mathtt{W}[2], f(2)\rangle; R$

$\mathit{OddInt(i)} \triangleq a[\mathtt{W}[2{*}i{+}1]](y).y!\langle \mathtt{W}[4{*}i{+}3], f(4{*}i{+}3)\rangle; y!\langle \mathtt{W}[4{*}i{+}4], f(4{*}i{+}4)\rangle; y?(\mathtt{W}[i], z); R'$

$\mathit{EvenInt(i)} \triangleq a[\mathtt{W}[2{*}i{+}2]](y).y!\langle \mathtt{W}[4{*}i{+}5], v_{2*i+2}\rangle; y!\langle \mathtt{W}[4{*}i{+}6], v'_{2*i+2}\rangle; y?(\mathtt{W}[i], z); Q'$

$\mathit{OddLeaf(i)} \triangleq a[\mathtt{W}[2*(2^{n-1}+i-1)+1]](y).y?(\mathtt{W}[2^{n-1}{+}i-1], z); S$

$\mathit{EvenLeaf(i)} \triangleq a[\mathtt{W}[2*(2^{n-1}+i-1)+2]](y).y?(\mathtt{W}[2^{n-1}+i-1], z); R$

$Tree \triangleq \lambda n.(\mathbf{R}\ (\mathbf{R}\ \mathit{Root}\ \lambda i.\lambda X.(\mathit{OddLeaf(i)} \mid \mathit{EvenLeaf(i)} \mid X)\quad 2^{n-1})$
$\lambda i.\lambda X.(\mathit{OddInt(i)} \mid \mathit{EvenInt(i)} \mid X)2^{n-1}{-}1)$

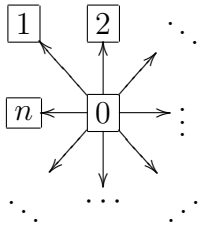
f is a function from natural numbers to U and W denotes the parameterised list of principals $\mathtt{W}[2]$, ..., $\mathtt{W}[2^{n+1}{-}2]$. The index calculation of children in *Root* is straightforward: $\mathtt{W}[1]$ and $\mathtt{W}[2]$. The index calculation of the roles that instantiate internal nodes is merely difficult; it directly follows the formulae in the global type once the expression that identifies them is defined, e.g. given the expression of principals for *OddInt* $2{*}i{+}1$, the expressions of

children and parent are trivial. However, the index calculation of the roles that instantiate leaves is rather complex: the formulae $(2 * (2^{n-1} + i - 1) + 1, 2 * (2^{n-1} + i - 1) + 2)$ that identify the principals in leaves, follow from a careful mathematical analysis on the range of principals in leaves, in relation to the index values generated from the **R** operator. *Root* is parameterised by the list of principals W at the shared name a, *OddInt* and *EvenInt* are parameterised by i, and *OddLeaf* and *EvenLeaf* are parameterised by i and n. In the main program, the inner **R** instantiates the leaves and root of a tree session whilst the outer one instantiates the internal nodes.

Common MPI error: Index Calculation The problem of index calculation in the roles of parallel algorithms has been recognised also by the MPI community (Gropp et al. (1999)) as a source of program errors. It is interesting to note that index calculation in the principals of global types is less complex than in those of roles. This is a direct advantage of the global representation of interactions where the information flow follows a straight line, mathematically represented through parametric linear functions. Our type system exploits this benefit of global types to type-check the indexes in roles. It statically ensures that the principals in the role's actions are the same to the ones specified in the causalities of global types; e.g. in the role *OddInt*, the type system ensures that the first message is sent to $\mathtt{W}[4{*}i{+}3]$ as it is correctly implemented in the role $(y!\langle 4{*}i{+}3, f(4{*}i{+}3)\rangle)$.

3.2.2 Star Pattern

The Star pattern consists of $n{+}1$ workers such that every worker $\mathtt{W}[\mathrm{i}]$ $(1 \leq i \leq n)$ is connected to $\mathtt{W}[0]$ as illustrated in Figure 3.2. The diagram explains the information flow of this pattern and the adjunct global type formalises it in our syntax.



R end
$\lambda j.\lambda \mathbf{y}.\mathtt{W}[0] \to \mathtt{W}[j+1] : \langle U \rangle.\mathbf{y}$
n

Figure 3.2: CPS: Diagram and global type of the Star pattern.

The global type specifies that the first message is sent by $\mathtt{W}[0]$ to $\mathtt{W}[n]$ for $j = n - 1$ and

the last one is sent by W[0] to W[1] for $n = 0$. The Star has two distinct roles: *Center*, represented by W[0], and *Worker*, represented by W[i]. The former sends a message to every connected worker and the latter receives that message:

$$\begin{array}{l}
\texttt{def}\ \ W = \mathbf{R}\ \texttt{W}[n]\ \lambda i.\lambda X.(\texttt{W}[i+2], X)\ n-2 \\
\mathit{Center} \triangleq \bar{a}[\texttt{W}[0], \texttt{W}[1], \vec{\texttt{W}}](y).\mathbf{R}\ \texttt{end}\ \lambda j.\lambda Y.y!\langle \texttt{W}[j+1], f(j+1)\rangle; Y\ n \\
\mathit{Worker(i)} \triangleq a[\texttt{W}[i+1]](y).y?(\texttt{W}[0], z); P \\
\text{Star} \triangleq \lambda n.(\mathbf{R}\ \mathit{Center}\ \lambda i.\lambda X.(\mathit{Worker(i)} \mid X) \quad n)
\end{array}$$

where f is a function from natural numbers to U and W denotes the parameterised list of principals W[2], ..., W[n]. *Center* is parameterised by n and *Worker* by i. In this pattern, we observe the use of the **R** operator inside the definition of the *Center* role to describe the repetitive behaviour of sending a message to all leaves. This example shows how to express iterative behaviour through the **R** operator in our system in a similar way as with the *for* loop. The typing of the role is given in Section 3.4.6. Lack of typing the *for* loop is a huge shortfall in SJ.

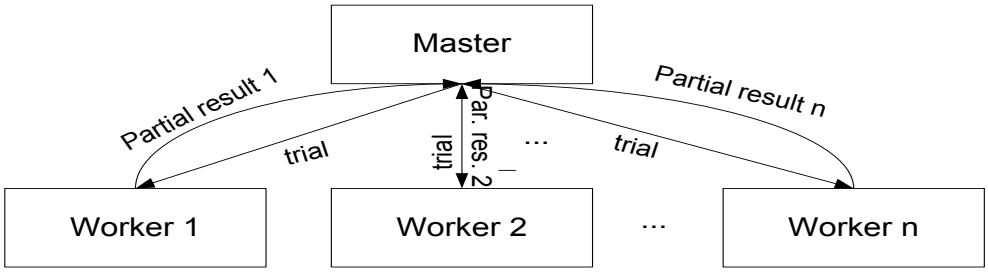


Figure 3.3: Monte Carlo: Diagram of information flow.

For example, the Monte Carlo algorithm computes a numerical approximation to the value of the constant π—ratio of any circle's circumference to its diameter. A Client process (or thread) sends the number of points to generate (trials) to a Master process and later receives from him the value of π. The Master divides the number of trials to the Worker(s) (trial) and instructs the latter to independently generate and check multiple sets of points in parallel, calculating the final value by combining the partial results from each Worker (see Figure 3.3). The code for a basic SJ implementation of the Worker and Master classes, typically present in the constructor of the class, looks like:

```
// Workers run the simulation.
final noalias SJSocket wm;
try (wm){
  // Accept the connection from Master.
  wm = ss.accept(); // sbegin
  int trial = wm.receive(); // ?(int)
  for (int i = 0; i < trial; i++)
    if (hit()) hits++;
  wm.send(hits); // !<int>
}catch
 (SJIncompatibleSessionException ise){
  System.err.println(...);
}
```

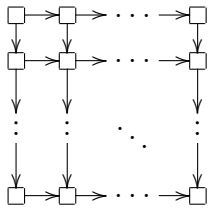
```
// Master controls the Workers.
final noalias SJSocket mw1, mw2, ...;
try (mw1, mw2, ...){
  mw1 = c_w2.request(); // cbegin
  mw2 = c_w3.request();
  ...
  // Multicast, !<int>
  <mw1, mw2, ...>.send(trial);
  // Collect the results, ?(int)
  int totalHits = mw1.receive()
       + mw2.receive() + ...;
  double pi = 4.0*((double)
      totalHits)/((double)trials);
```

where `mw1` is the Master's session socket to Worker1, and similarly for the other numbered `mw` variable; `wm` is the Worker's session socket with the Master. The method `hit` in Worker returns the Boolean from testing a generated point: true, if a point falls inside the circle, false otherwise. The Master can then calculate π by `4*totalHits / (trials)`, where `trials = trial*n` and `n` is the number of Workers. The dot syntax (...) in the multicast "send" in Master is an abusive, not a language construct, notation, denoting a list of sockets. The multicast send construct is not parameterised by the number of workers and so, we represent the list using the ... notation. In reality, for every instance of workers, a new Master class is written in SJ, containing the specific multicast send for that number of workers. SJ lacks parameters in session types, prohibiting programmers from representing multiple computations in a single class. This restriction results in unscalable programs. Our system provides a model that supports the use of *for* in programs, increasing expressivity and ensuring the correctness of the them through the type system.

3.2.3 Mesh Pattern

The Mesh pattern organises workers in a Grid. The diagram in Figure 3.4 describes communications over a two dimensional mesh pattern (2D-Mesh) of size $n \times m$ where parameters n and m represent the number of rows and the columns. Variants of this pattern include toric meshes and hypercubes. In the adjunct global type, the two causalities $\mathtt{W}[i+1][j+1] \rightarrow \mathtt{W}[i][j+1] : \langle \mathsf{nat} \rangle$ and $\mathtt{W}[i+1][j+1] \rightarrow \mathtt{W}[i+1][j] : \langle \mathsf{nat} \rangle$ specify the fact that each worker not situated on the last row or column sends a message to his

neighbors situated below and on its right. The causalities $\mathtt{W}[i+1][0] \to \mathtt{W}[i][0] : \langle \mathsf{nat} \rangle$ and $\mathtt{W}[0][k+1] \to \mathtt{W}[0][k] : \langle \mathsf{nat} \rangle$ describe the messages sent in the last column and row.



$$
\begin{array}{l}
(\mathbf{R}\ \ (\mathbf{R}\ \mathsf{end}\ \lambda k.\lambda \mathbf{z}.\mathtt{W}[0][k+1] \to \mathtt{W}[0][k] : \langle \mathsf{nat} \rangle.\mathbf{z}\ m) \\
\quad \lambda i.\lambda \mathbf{x}. \\
\quad\ (\mathbf{R}\ (\mathtt{W}[i+1][0] \to \mathtt{W}[i][0] : \langle \mathsf{nat} \rangle.\mathbf{x}) \\
\qquad \lambda j.\lambda \mathbf{y}. \\
\qquad\ \ \mathtt{W}[i+1][j+1] \to \mathtt{W}[i][j+1] : \langle \mathsf{nat} \rangle. \\
\qquad\ \ \mathtt{W}[i+1][j+1] \to \mathtt{W}[i+1][j] : \langle \mathsf{nat} \rangle.\mathbf{y} \\
\quad\ m) \\
\quad n)
\end{array}
$$

Figure 3.4: CPS: Diagram and global type of the Mesh pattern.

The roles of the pattern are defined by the communication behaviour and by the links with neighbors of each worker. 2D-Mesh has six kinds of communication behaviour: 1) two "send" (top left corner), 2) one "send" and one "receive" (top right, bottom left corners) 3) two "receive" (bottom right corner), 4) two "send" and one "receive" (top row, left column), 5) two "receive" and one "send" (bottom row, right column) 6) two "send" and two "receive" (internal nodes). From the diagram, it is easy to identify that the links between the principals of the top row and left column are different, resulting in two different roles; e.g., the receiving worker for top row is $\mathtt{W}[n][i+2]$ while for the left column it is $\mathtt{W}[i+2][m]$. Also, the links between bottom row and right column are different, therefore two different roles, concluding nine roles for the Mesh pattern. For presentation reasons, we relegate the definition of roles and main program in Appendix B.2.1.

3.3 Real-World Examples

The communication patterns discussed in the previous section are studied in concrete real world examples, namely, Jacobi Solution of the Poisson Equation, Dense Linear Algebra and Group Diffie-Hellman with Complete Key Authentication Protocol, illustrating the practical utility of our system.

3.3.1 Jacobi Solution of the Discrete Poisson Equation

Poisson's equation is a partial differential equation with applications in heat flow, electrostatics, gravity and climate computations. The discrete two-dimensional Poisson equation

$(\nabla^2 u)_{ij}$ for a $m \times n$ grid u is

$$(\nabla^2 u)_{ij} = \frac{1}{dx^2}(u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{ij}) = g_{ij}$$

where $2 \leq i \leq m-1$ and $2 \leq j \leq n-1$. The value of each element of the matrix (representing mathematically the grid) u can be expressed as

$$u_{ij} = \frac{1}{4}(u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - dx^2 g_{ij})$$

where $1 \leq i \leq m-2$, $1 \leq j \leq n-2$, and $dx = 1/(n+1)$. Jacobi's Method or *Jacobi* as we will refer to it throughout this thesis converges on a solution by repeatedly replacing each element of the matrix u by an average of its four neighbouring values and $dx^2 g_{i,j}$; for this example, we set g to 0. Then from the k-th approximation of u, the next iteration calculates:

$$u_{ij}^{k+1} = \tfrac{1}{4}(u_{i+1,j}^k + u_{i-1,j}^k + u_{i,j+1}^k + u_{i,j-1}^k)$$

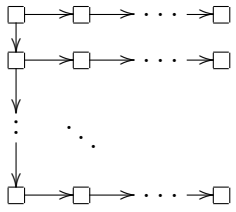
The code for this calculation, taking $(\nabla^2 u)_{ij} = 0$, is:

```
for(int i = 1; i < m - 1; i++)
  for(int j = 1; j < n - 1; j++)
    newu[i][j] = (u[i-1][j] + u[i+1][j] + u[i][j-1] + u[i][j+1]) / 4.0;
```

Each iteration of the algorithm involves O(mn) computation steps (the number of elements in the matrix). Termination may be on reaching a target convergence threshold or completing a certain number of iterations. The Jacobi solution enables parallelisation by exploiting the fact that each element of the grid can be updated independently (within one step): the grid can be divided up and the algorithm performed on each subgrid in separate processes or threads. The key to success is that neighbouring processes must exchange their subgrid boundary values as they are updated. We present a parameterised version of it over a two-dimensional grid (mesh). The grid is divided into $n{*}m$ processes, where $n, m{\geq}2$. In addition to their allocated subgrid, each process maintains a copy of the boundary values (*ghost points*) of its neighbours; the new values are communicated after each iteration. The process on the (n, m) subgrid, top left corner (see diagram below), controls the termination condition for all processes and so, sends the first message in the mesh. The global type for the said interactions is:

$$\mathit{Jacobi} \triangleq \mu \mathbf{t}.\mathtt{W}[n][m] \to \mathtt{W}[n][m-1], \mathtt{W}[n-1][m] : \{\mathtt{true} : \mathit{iterate}, \mathtt{false} : \mathit{return}\}.$$

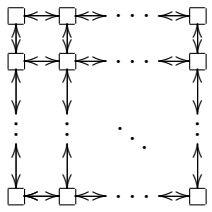
where *iterate* and *return* denote global types that will be considered in the following paragraphs. The stopping condition is propagated in the processes following the pattern of the diagram below. Next to it, we present the global type (*iterate*) for propagating the `true` label.



1. **R**
2. **R** (**R** *ghst–pnt* $\lambda k.\lambda\mathbf{z}.\mathtt{W}[0][k{+}1] \to \mathtt{W}[0][k]{:}\{\mathtt{true}{:}\mathbf{z}\}$ m)
3. $\lambda i.\lambda\mathbf{x}.$
4. (**R** $\mathtt{W}[i{+}1][m] \to \mathtt{W}[i][m]{:}\{\mathtt{true}{:}\mathbf{x}\}$
5. $\lambda j.\lambda\mathbf{y}.\mathtt{W}[i{+}1][j{+}1] \to \mathtt{W}[i{+}1][j]{:}\{\mathtt{true}{:}\mathbf{y}\}$
6. m)
7. $n{-}1$
8. $\lambda l.\lambda\mathbf{w}.\mathtt{W}[n][l{+}1] \to \mathtt{W}[n][l]{:}\{\mathtt{true}{:}\mathbf{w}\}$ $(m{-}1)$

Propagation of the label in the top row, is described in the causality of line 8, in all the rows, except top and bottom, line 5, in the leftmost column in line 4 and in the bottom row, line 2.

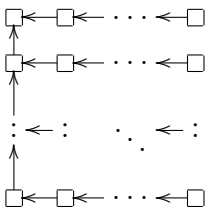
Each process maintains a copy of the boundary values of its neighbours and exchanges them on each iteration of the algorithm. The diagram below portrays how these values are exchanged between the processes, followed by the global type (*ghst–pnt*).



1. **R** (**R** *conv–data* $\lambda k.\lambda\mathbf{z}.\mathtt{W}[0][k+1] \leftrightarrow \mathtt{W}[0][k]{:}\langle U\rangle$ m)
2. $\lambda i.\lambda\mathbf{x}.$
3. (**R** $\mathtt{W}[i+1][0] \leftrightarrow \mathtt{W}[i][0]{:}\langle U\rangle.\mathbf{x}$
4. $\lambda j.\lambda\mathbf{y}.\mathtt{W}[i+1][j+1] \leftrightarrow \mathtt{W}[i][j+1]{:}\langle U\rangle.$
5. $\mathtt{W}[i+1][j+1] \leftrightarrow \mathtt{W}[i+1][j]{:}\langle U\rangle.\mathbf{y}$
6. m)
7. n

$\mathtt{p} \leftrightarrow \mathtt{p}'{:}\langle U\rangle$ is a shortcut for $\mathtt{p} \to \mathtt{p}'{:}\langle U\rangle.\mathtt{p}' \to \mathtt{p}{:}\langle U\rangle$. The exchange of ghost-points in all the rows and columns, except the rightmost column line 3 and bottom row line 1, is described in the causalities of line 4 and 5.

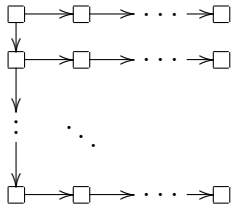
The convergence data are gathered at the root processes following the pattern of the diagram below and next to it, the global type (*conv-data*).



1. **R**
2. (**R t** $\lambda k.\lambda \mathbf{z}.\mathtt{W}[n][m-k-1] \rightarrow \mathtt{W}[n][m-k] : \langle U' \rangle.\mathbf{z}\ m$)
3. $\lambda i.\lambda \mathbf{x}.$
4. (**R** ($\mathtt{W}[n-i-1][m] \rightarrow \mathtt{W}[n-i][m] : \langle U' \rangle.\mathbf{x}$)
5. $\lambda j.\lambda \mathbf{y}.$
6. $\mathtt{W}[n-i-1][m-j-1] \rightarrow \mathtt{W}[n-i-1][m-j] : \langle U' \rangle.\mathbf{y}$
7. m)
8. n

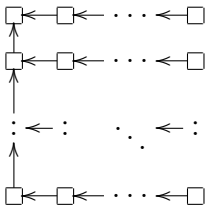
The data is firstly collected from all the rows in the first element of them as described in the causality in line 6. However, the collection of data in the first row is described through the causality in line 2. Secondly, the data collected in the first column is then forwarded into the top right corner element, coordinates (n, m), line 4.

The `false` label is propagated similarly as the `true` label (*iterate*). The causalities of this global type (*return*) are the same as in the *iterate* global type.



1. **R**
2. **R** (**R** *result* $\lambda k.\lambda \mathbf{z}.\mathtt{W}[0][k{+}1] \rightarrow \mathtt{W}[0][k]{:}\{\mathtt{false}{:}\mathbf{z}\}\ m$)
3. $\lambda i.\lambda \mathbf{x}.$
4. (**R** $\mathtt{W}[i{+}1][m] \rightarrow \mathtt{W}[i][m]{:}\{\mathtt{false}{:}\mathbf{x}\}$
5. $\lambda j.\lambda \mathbf{y}.\mathtt{W}[i{+}1][j{+}1] \rightarrow \mathtt{W}[i{+}1][j]{:}\{\mathtt{false}{:}\mathbf{y}\}$
6. m)
7. $n{-}1$
8. $\lambda l.\lambda \mathbf{w}.\mathtt{W}[n][l{+}1] \rightarrow \mathtt{W}[n][l]{:}\{\mathtt{false}{:}\mathbf{w}\}\ m{-}1$

The partial results are gathered at the root processes following the pattern of the diagram below and next to it, the global type (*result*).



1. **R**
2. (**R** end $\lambda k.\lambda \mathbf{z}.\mathtt{W}[n][m-k-1] \rightarrow \mathtt{W}[n][m-k] : \langle Grid \rangle.\mathbf{z}\ m$)
3. $\lambda i.\lambda \mathbf{x}.$
4. (**R** ($\mathtt{W}[n-i-1][m] \rightarrow \mathtt{W}[n-i][m] : \langle Grid \rangle.\mathbf{x}$)
5. $\lambda j.\lambda \mathbf{y}.$
6. $\mathtt{W}[n-i-1][m-j-1] \rightarrow \mathtt{W}[n-i-1][m-j] : \langle Grid \rangle.\mathbf{y}$
7. m)
8. n

The causalities in this global type are the same as the ones in *conv-data*, except the type of the messages sent, which in this global type is Grid— solution on each component of the decomposed initial grid. For presentation reasons, we relegate the definition of roles

into Appendix B.2.2.

3.3.2 Dense Linear Algebra

The Dense Linear Algebra family of algorithms consists of the Matrix-Vector and Matrix-Matrix multiplication algorithms. The product of a matrix with a vector or another matrix is computed following the Gaussian method. That is, the product of an $n \times m$ matrix A with an m vector B is

$$C_i = \sum_{j=0}^{m} A_{ij} B_j \qquad (1)$$

where $0 \leq i \leq n$ is the row index and the product of an $n \times m$ matrix A with an $m \times r$ matrix B is

$$C_{ij} = \sum_{k=0}^{m} A_{ik} B_{kj} \qquad (2)$$

where $0 \leq i \leq n$ is the row index and $0 \leq j \leq r$ is the column index.

The nature of matrix product is amenable to parallelism. A Master process divides the matrix in rows and distributes them to the Worker(s) and so, the latter multiplies the rows with the other factor of the product: vector or matrix, depending on the algorithm. The final result is calculated by gathering the partial results from each Worker. The information flow of these algorithms is designed in a Star pattern, similarly to the Monte Carlo one. We use our recursive construct to model the scattering of the matrix to the Workers in a similar way as using the *for* loop. The matrix is divided into n sub-matrices by the Master (Center role) and each piece is sent to a particular Worker, where n is the number of workers; Worker(s) compute the partial result of the multiplication and send it to the Master. The fore-mentioned operations are preceded by the broadcast of the second product factor: vector, in the case of matrix-vector multiplication, or matrix, in the case of matrix-matrix multiplication. The global type of this session is:

1. $\mathbf{R}$
2. $(\mathbf{R}\ \mathsf{end}\ \lambda i.\lambda \mathbf{x}.Master \rightarrow \mathtt{W}[i] : \langle int[][] \rangle.\mathtt{W}[i] \rightarrow Master : \langle U \rangle.\mathbf{x}\ n)$
3. $\lambda j.\lambda \mathbf{z}.$
4. $Master \rightarrow \mathtt{W}[i] : \langle U \rangle.\mathbf{z}$
5. n

where the causality in line 4 describes the broadcast of the second factor to all the worker, U denotes an array of integers ($int[]$) in case of the matrix-vector multiplication or a matrix of integers ($int[][]$) in case of the matrix-matrix multiplication; the causalities in line 2 describe scattering of the matrix and then gathering of the partial results from Worker(s) to Master.

The Master role presents the most interesting case for this study. Below, we present the definition of Master:

$$\begin{aligned}\bar{a}[\mathtt{W}[0], \mathtt{W}[1], \mathtt{W}](y).\ \mathbf{R}\\ &(\mathbf{R}\ \mathsf{end}\ \lambda i.\lambda \mathbf{x}.y!\langle \mathtt{W}[i], a[i]\rangle; y?\langle \mathtt{W}[i], x[i]\rangle; \mathbf{x}\ n)\\ &\lambda j.\lambda \mathbf{z}.y!\langle \mathtt{W}[j], b\rangle; \mathbf{z}\\ &n\end{aligned}$$

where W is defined in Section 3.2.2 for the Star pattern. The outer $\mathbf{R}$ sends the second factor (b) in broadcast to every worker of the session and the inner $\mathbf{R}$ distributes the matrix over each worker and receives the results of the multiplication between the sub-matrices and the second factor. In this role, $a[i]$ denotes the ith decomposition of the matrix a and $x[i]$ denotes the result associated to the ith submatrix.

In SJ, we overcome the challenge of having an arbitrary number of Worker(s), passed as a program parameter to the Master through the command line, using threads. SJ captures interactions between only two participants and so, does not capture interactions between an arbitrary number of participants. Given the number of Worker(s), the Master's behaviour is to generate MasterWorker threads— the real components of this application. Below, we show how the Master creates and runs those threads:

```
MasterWorkerThread threads[] = new MasterWorkerThread[NUM_WORKERS];
for(int i = 0; i < NUM_WORKERS; i++){
    threads[i] = new MasterWorkerThread(a, b, SIZE, result, index_row, lock,
        hosts_w[i], ports_w[i]); /* Matrix, vector and variables used to access
        the monitor and set up the communication with a Worker */
    threads[i].start();
}
```

Each thread has references to factors of the multiplication, variables to synchronise the different threads and variables to set up the communication. The threads use a monitor to synchronise the accesses to the main matrix A when selecting the next available row to send for multiplication. However, this solution carries a cost, coming from the creation

and synchronisation of threads, that decreases the efficiency of the algorithm. While in CPS, we use the **R** operator to express the iterative behaviour of Master in relation to the arbitrary number of participants in a single process (thread) and type-check it with the global type.

3.3.3 Group Diffie-Hellman with Complete Key Authentication Protocol

The Diffie-Hellman protocol is used in password-authentication key agreement and public key infrastructure. The protocol of authenticated key agreements is studied in dynamic peer groups by Ateniese et al. (1998). Every group M_i ($0<i<n$) generates and encrypts a random exponent that, together with the data received from M_{i-1}, is then sent to M_{i+1}. Lastly, M_n receives the data from M_{n-1}, computes the group key and broadcasts it to all other parties. The global type of the protocol is defined as:

$$\begin{array}{ll} 1. & \mathbf{R}\ \ (\mathbf{R}\ \mathsf{end}\ \lambda k.\lambda \mathbf{z}.\mathtt{M}[n] \rightarrow \mathtt{M}[k]\colon \langle key\rangle.\mathbf{z}\ n) \\ 2. & \quad \lambda i.\lambda \mathbf{x}.\mathtt{M}[n-i-1] \rightarrow \mathtt{M}[n-i]\colon \langle data\rangle.\mathbf{x}\ n \end{array}$$

where the causality in line 1 describes broadcasting of the group key to every worker and the causality in line 2 describes the exchange of data through the workers organised over a line pattern. The session is defined over three roles as in the Ring pattern: Starter, Middle and Last. Below, we give the definition of the three roles in CPS.

$$\begin{array}{l} \mathit{Starter} \triangleq \bar{a}[\mathtt{M}[0], \mathtt{M}[1], \tilde{\mathtt{M}}](y).y!\langle \mathtt{M}[1], d\rangle; y?(\mathtt{M}[n], z); R \\ \mathit{Middle(i)} \triangleq a[\mathtt{M}[i+1]](y).y?(\mathtt{M}[i], x); y!\langle \mathtt{M}[i+2], f(x)\rangle; y?(\mathtt{M}[n], z); R' \\ \mathit{Last} \triangleq a[\mathtt{M}[n]](y).y?(\mathtt{M}[n-1], x); \mathbf{R}\ S\ \lambda k.\lambda \mathbf{z}.y!\langle \mathtt{M}[k], key\rangle; \mathbf{z}\ n \end{array}$$

where $\tilde{\mathtt{M}}$ denotes the list $\mathtt{M}[2]$, ..., $\mathtt{M}[n]$. The main program is defined similarly as in Section 3.1.

This protocol is modelled in a system of contracts by Castagna and Padovani (2009). In that model, an extra private channel is used by the last group M_n to send the key to every other group. The private channel is forwarded between the groups through delegation. A condition is added to the protocol description to check whether the key is sent to every group or not.

$P_0 = c^1!\langle d\rangle.c^1?\langle y\rangle.y \oplus ok.\mathbf{0}$
$P_i = c^i?(x : data).c^{i+1}!\langle f(x)\rangle.c^{i+1}?\langle y\rangle.y \oplus key.y!\langle K_i\rangle.y?(s : Int).c^i!\langle y\rangle.\mathbf{0}$
$P^n = c^n?(x : data).c^n!(c).Q$ with $Q = c\&\{ok : \mathbf{0}, key : c?(k : Int).c!\langle g(x, k)\rangle.Q$

The most interesting process is the last one, P_n. After receiving the data from a neighbor, it sends the private channel c and enters into a repetitive behaviour. In this state, P_n checks the identity of the neighbor through the stopping condition, making sure the neighbor is P_0 and thereby ending the session; otherwise, it sends the group key to the neighbor given the label K_i that identifies the neighbor.

In our model, we do not need an extra private channel c to send the key, as communications between parties of a session are always defined over private channels; we do not need to specify a special message between P_i and P_n that contains a label K_i to identify a worker inside a group as this is granted by the syntax of the **R** operator (index i); also, we do not need to add a condition that checks whether the key is sent to every group as this is granted by the semantics of the **R** operator.

3.4 Formal Model

Our system is modelled on that of Bettini et al. (2008), where channels are omitted from the syntax of roles and global types, serving a simpler type system than the one introduced by Honda et al. (2008b)— no definition of linearity. Our syntax of global types and roles extends the previous one with parameterised principals and the primitive recursive operator **R** from Gödel's System T.

Primitive recursion forms a new kind of global type— product kind that abstracts all instances of a product global type. We introduce well-formedness rules, in the form of a kind system, that ensure in a global type that indexed principals range over natural numbers and parametric expressions are applied only to product global types. CPS preserves the programming methodology and typing strategy of multiparty session types: programmers first define the global type of the intended pattern and then define each role of it, and roles are validated by type-checking through (1) projection of the global type onto the principals and (2) sorting and **R**-elimination of role types.

For simplicity of the theory, including proofs, we have omitted multicast send of values and labels, and higher-order communication from this study.

3.4.1 Syntax

Global types Figure 3.5 gives the syntax of global types, including the new constructs added to CSMS (highlighted in grey). The metavariable $\mathtt{p}$ (as well as with suffix and subscript) ranges over principals; $\mathtt{i}$ (as well as with subscript) ranges over mathematical expressions containing an index (special variable), hence, index expressions; $\mathtt{t}$ (as well as with subscript) ranges over mathematical expressions containing a parameter (special variable), hence, parametric expressions; $\mathcal{N}$ ranges over participants. The other metavariables read the same as in CSMS.

In contrast to CSMS, the message exchange and branching constructs $\mathtt{p} \to \mathtt{p}' : \langle \tilde{U} \rangle$, $\mathtt{p} \to \mathtt{p}' : \{l_i : G_i\}_{i \in I}$ do not contain channels in causalities. However, they read similarly as recursion and type variable: infinite behaviour is represented by recursively defined global types $\mu \mathbf{t}.G$.

The construct $\mathbf{R}\ G\ \lambda i.\lambda \mathbf{x}.G'$ captures communication patterns of an arbitrary number of principals. The $\mathbf{R}$ operator is defined over the tail recursion case of recursion and index variable i. Thus the operator preserves the existing declarative nature of global types, allowing that the system developed in our model can be extended with additional features. The parameters that abstract the number of participants are bound by the binders in the lambda expressions of roles, since both global type and roles are part of the program definition. Throughout the dissertation we will refer to primitive recursive global types as *product global types*, as they abstract all instances of the parameterised global type. Although, we do not have a formal demonstration, we believe that the $\mathbf{R}$ operator cannot be encoded in the syntax of multiparty global types with parameters. The μ operator in that syntax —the only construct that models iteration— models the repetitive behaviour of a group of participants and not the repetitive behaviour of a communication pattern, defined over a parameterised group of participants. So, additional constructs are needed in the syntax of multiparty global types to model the behaviour of a parameterised group of participants. We leave the investigation of such constructs as future work.

The infinite set of instances generated from the $\mathbf{R}$ operator can be understood through the two reduction rules, provided by the construct $G\ \mathtt{t}$, given in Section 1.5. `end` signifies the end of a conversation and is used as a base type to build more complex global types.

Principals $\mathtt{p}, \mathtt{p}', \mathtt{q}, ..$ include primitive participants `Alice`, `Bob`, ... and indexed principals defined over one or multiple index expressions $\mathtt{W}[i]$, $\mathtt{W}[i{+}1][j{+}1]$, ... Index expressions $\mathtt{i}$ are represented by parametric linear functions, where n ranges over natural numbers, i ranges

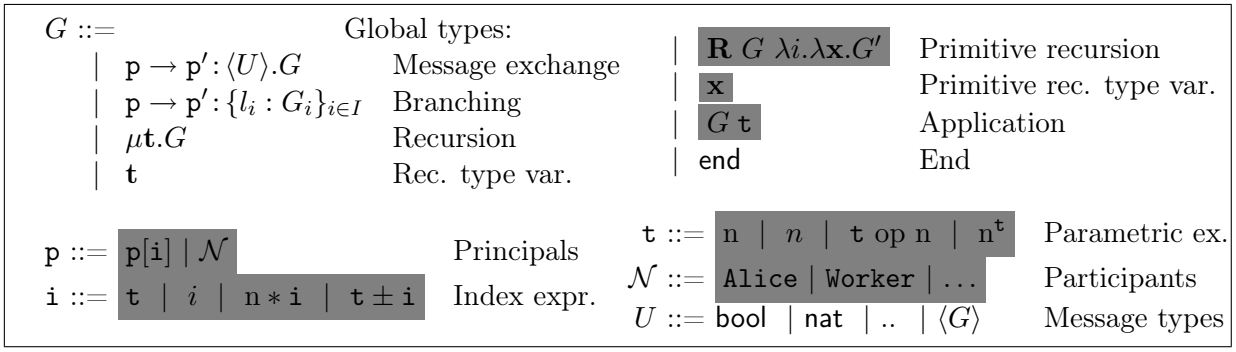


Figure 3.5: CPS: Syntax of global session types.

over index variables and $\mathtt{t}$ ranges over parametric expressions. Parametric expressions range over variables n, natural numbers n, arithmetical operations ($\mathtt{t}+\mathrm{n}$, $\mathtt{t}-\mathrm{n}$, $\mathtt{t}*\mathrm{n}$) and exponentiation of base natural number. Our design of index expressions as parametric linear functions comes from the observation that the information flow follows a straight line, neither a curve nor other forms of line, in the patterns/virtual-topologies we have studied so far, namely Ring, Star, Tree and Mesh. The observation follows from the fact that communication in mobile processes is defined between two discrete end-points and not between an infinite number of end-points (the study of communications over an infinite number of end-points is an interesting mathematical problem worth investigating). For simplicity and without reducing the practical expressiveness of our system, we have designed index expressions to have at most one parameter n. A type U ranges over primitive (bool, nat, ...) and global types ($\langle G \rangle$), excluding role types to prohibit higher-order communication.

Roles and processes Figure 3.6 provides the syntax of programs, and new constructs that make roles and processes (in grey). The metavariable E stands for general expressions; R (as well as with subscript and suffix) stands for roles and processes when the former is defined over principals and the latter over value principals. The variable p is a metavariable ranging over lists of principals; h ranges over queues; m ranges over messages; $\hat{\mathtt{p}}$ (as well as $\hat{\mathtt{q}}$) ranges over value principals. The other metavariables read the same as in CSMS.

A program $\lambda n.E$ is a function from natural numbers n (the number of participants) to roles (R) in E composed in parallel and it is invoked by applying natural numbers through the construct $E\ \mathtt{t}$.

Roles in our calculus are second-class constructs; they cannot be computed by functions. Each role defines a scope that includes the subsequent behaviours. The role with the

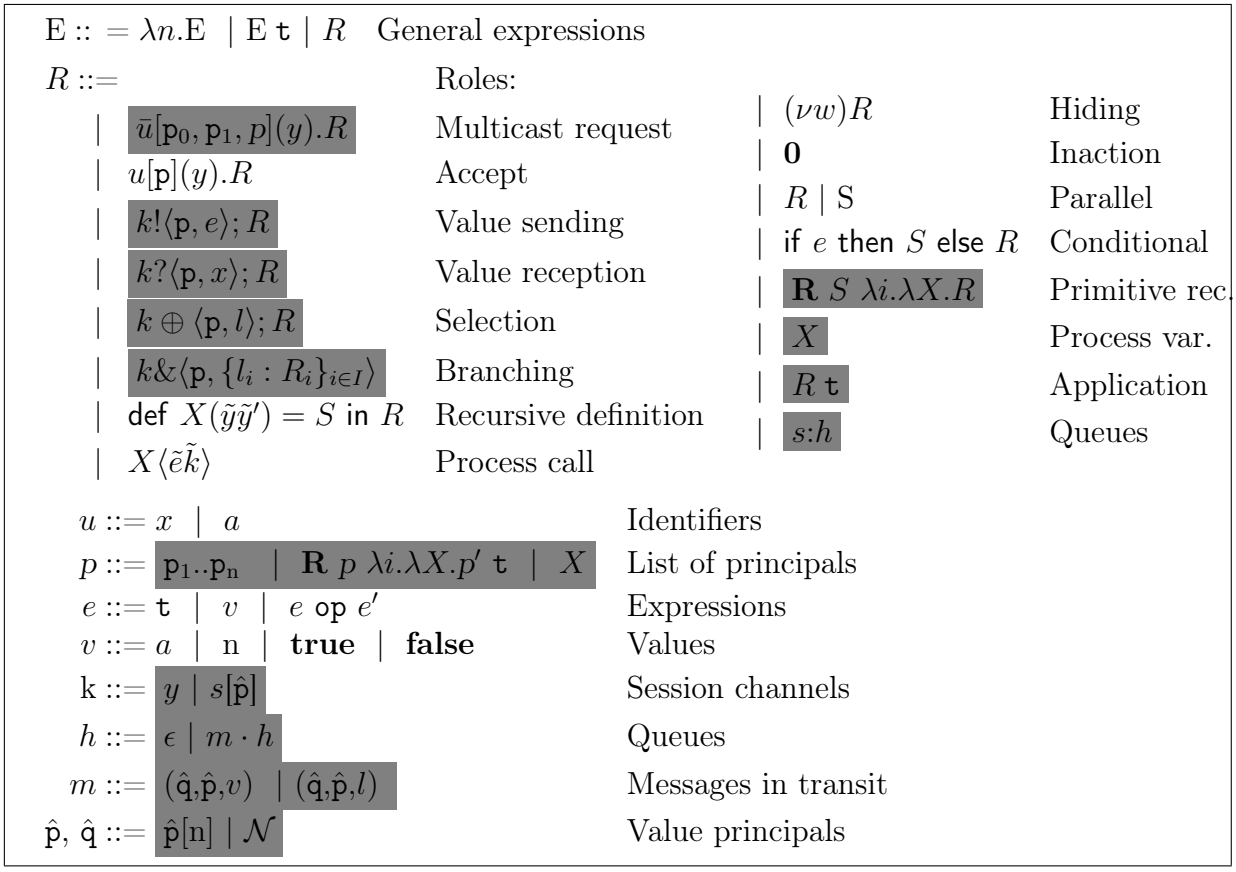


Figure 3.6: CPS: Syntax of general expressions, roles and processes.

overbar $\bar{u}$ prefix $\bar{u}[\mathtt{p}_0, \mathtt{p}_1, p](y).R$ represents the behaviour of the first principal in the list (possibly parameterised) of principals $\mathtt{p}_0, \mathtt{p}_1, p$ and the process of that role initiates a session with the acceptor processes of principals $\mathtt{p}_1$ and p. The role $u[\mathtt{p}](\tilde{x}).R$ accepts a request from the Request process where the session identifier u serves as a public point of access for all the participants to dynamically set the session channels (place-held by $\tilde{x}$) that perform the actions defined in R.

In sending and receiving constructs $k!\langle \mathtt{p}, e\rangle; R$ and $k?\langle \mathtt{p}, x\rangle; R$, the principal $\mathtt{p}$ denotes the other end-point of the communication; the same notation is used in selection-branching of a label $k \oplus \langle \mathtt{p}, l\rangle; R$ and $k\&\langle \mathtt{p}, \{l_i : R_i\}_{i\in I}\rangle$ where the former selects one of the labels enumerated in I and sends it to the later. This representation of communication primitives is a consequence of the Bettini *et al.* type system that represents global types without channels (in that setting, channels are considered as unshared entities).

The construct def D in R defines infinite behaviour in the calculus through recursive functions, instantiated through the $X\langle \tilde{e}\tilde{k}\rangle$ construct, typically present in the scope of R;

hiding is standard, restricting the scope of actions of shared names (a) and session channels (s) only to R; $\mathbf{0}$ describes the end of a behaviour; conditional has the same definition as in imperative languages; the parallel construct is standard, composing in parallel the behaviour of two processes.

The $\mathbf{R}$ operator in the $\mathbf{R}\ S\ \lambda i.\lambda X.R$ construct parameterises principals, iterates and composes in parallel roles. The recursive operator can be used also inside the definition of a role to iterate over a particular end-point behaviour and to sequentially compose behaviours as shown in Sections 3.2.2 and 3.3.2. Another use of $\mathbf{R}$ is to define lists of principals of different length (p). Iteration takes place when a natural number is applied to a primitive recursion term through construct $R\ \mathtt{t}$.

The message queue is part of the runtime syntax of the calculus. The message queue can be visualised as the Internet: the set of machine memories, cables, routers and other network hardware, where messages are stored before the application level reads them. Identifiers of a session include variables and shared names. A list of principals can be constant or parameterised using the $\mathbf{R}$ operator. Expressions include parametric mathematical expressions, values and operations such as $e = e'$, e and e' and not e. Values are defined over shared names, natural numbers and boolean values; session channels are not allowed in the syntax of values to avoid higher-order communication in the calculus. Channels denote channel variables or session channels; session channels are labeled by the principal identifier to identify a participant in a session as in CSMS. Messages in queues are defined as triples: sender, receiver and data (value or label) just as IP packets. Messages are run-time entities, therefore they are defined over value principals. Value principals include participants (`Bob`, `Alice`, ...) or indexed principals over natural numbers (`W`[3], `W`[2][4], ...). In the case when value principals are used instead of parameterised principals in the syntax of roles, we obtain the syntax of processes.

Definition 3.4.1. The set of free names of a general expression E, written $fn(\mathrm{E})$, is defined as follows:

$$
\begin{aligned}
fn(\lambda n.E) &= fn(\mathrm{E}) & fn(\mathsf{if}\ e\ \mathsf{then}\ S\ \mathsf{else}\ R) &= fn(\mathrm{S}) \bigcup fn(R)\\
fn(\mathrm{E}\ \mathtt{t}) &= fn(\mathrm{E}) & fn(R \mid \mathrm{S}) &= fn(R) \bigcup fn(\mathrm{S})\\
fn(\bar{u}[\mathtt{p}_0, \mathtt{p}_1, p](y).R) &= fn(u) \bigcup fn(R) & fn(u[\mathtt{p}](y).R) &= fn(u) \bigcup fn(R)\\
fn(k!\langle \mathtt{p}, e\rangle; R) &= fn(e) \bigcup fn(R) & fn(X\langle \tilde{e}\tilde{k}\rangle) &= fn(\tilde{e})\\
fn(k?\langle \mathtt{p}, x\rangle; R) &= fn(R) & fn(\mathbf{R}\ S\ \lambda i.\lambda X.R) &= fn(\mathrm{S}) \bigcup fn(R)\\
fn(k \oplus \langle \mathtt{p}, l\rangle; R) &= fn(R) & fn(X) &= \emptyset\\
fn(k\&\langle \mathtt{p}, \{l_i : R_i\}_{i\in I}\rangle) &= \textstyle\bigcup_{i\in I} fn(R_i) & fn(R\ \mathtt{t}) &= fn(R)
\end{aligned}
$$

$$fn(\mathbf{0}) = \emptyset \qquad fn(\mathsf{def}\ X(\tilde{y}\tilde{y}') = S\ \mathsf{in}\ R) = fn(\mathrm{S}) \bigcup fn(R)$$
$$fn((\nu w)R) = fn(R) \setminus fn(w) \qquad fn(s{:}h) = fn(h)$$
$$fn(\mathrm{x}) = \emptyset \quad fn(a) = \{a\} \quad fn(s[\hat{\mathtt{p}}]) = \emptyset \quad fn(\mathrm{n}) = \emptyset \quad fn((\hat{\mathtt{q}},\hat{\mathtt{p}},v)) = fn(v)$$
$$fn(\mathrm{X}(\tilde{y},\tilde{y}')\mathrm{=S}) = fn(\mathrm{S})$$

Definition 3.4.2. The set of free channels of a general expression E, written $fc(\mathrm{E})$, is defined as follows:

$$fc(\lambda n.E) = fc(\mathrm{E}) \qquad fc(\mathsf{if}\ e\ \mathsf{then}\ S\ \mathsf{else}\ R) = fc(\mathrm{S}) \bigcup fc(R)$$
$$fc(\mathrm{E}\ \mathtt{t}) = fc(\mathrm{E}) \qquad fc(R \mid \mathrm{S}) = fc(R) \bigcup fc(\mathrm{S})$$
$$fc(\bar{u}[\mathtt{p}_0,\mathtt{p}_1,p](y).R) = fc(R) \qquad fc(u[\mathtt{p}](y).R) = fc(R)$$
$$fc(k!\langle \mathtt{p}, e\rangle; R) = fc(\mathrm{k}) \bigcup fc(R) \qquad fc(X\langle \tilde{e}\tilde{k}\rangle) = fc(\tilde{k})$$
$$fc(k?\langle \mathtt{p}, x\rangle; R) = fc(\mathrm{k}) \bigcup fc(R) \qquad fc(\mathbf{R}\ S\ \lambda i.\lambda X.R) = fc(\mathrm{S}) \bigcup fc(R)$$
$$fc(k \oplus \langle \mathtt{p}, l\rangle; R) = fc(\mathrm{k}) \bigcup fc(R) \qquad fc(X) = \emptyset$$
$$fc(R\ \mathtt{t}) = fc(R)$$

$$fc(\mathbf{0}) = \emptyset \qquad fc(\mathsf{def}\ X(\tilde{y}\tilde{y}') = S\ \mathsf{in}\ R) = fc(\mathrm{S}) \bigcup fc(\mathrm{R})$$
$$fc((\nu w)R) = fc(R) \setminus fc(w) \qquad fc(s{:}h) = \emptyset$$

$$fc(k\&\langle \mathtt{p}, \{l_i : R_i\}_{i\in I}\rangle) = \bigcup_{i\in I} fc(R_i) \bigcup fc(\mathrm{k})$$
$$fc(\mathrm{x}) = \emptyset \quad fc(a) = \emptyset \quad fn(s[\hat{\mathtt{p}}]) = \{s[\hat{\mathtt{p}}]\} \quad fc(\mathrm{n}) = \emptyset \quad fc(\mathrm{X}(\tilde{y},\tilde{y}')\mathrm{=S}) = fc(\mathrm{S})$$

3.4.2 Operational Semantics

Figure 3.7 gives the operational semantics via the reduction relation $\longrightarrow$, written $E \longrightarrow E'$, read "program E reduces to program E' in one step". The interesting features of the rules are how they invoke a program, start a session, instantiate roles, iterate over end-point behaviour and exchange messages.

The rule [App] invokes a program by replacing the parameter n with the argument n, as it instantiates roles which are parameterised only by n. The features of rules [Zero] and [Succ] were discussed in Section 3.1. However, it is of interest to mention in the two rules that when R denotes roles, [Succ] instantiates them in each iteration and composes them in parallel. Otherwise R denotes an end-point behaviour that [Succ] iterates when the session has been established.

A session is established among processes via shared channels as in CSMS. At this point, every role has been instantiated into processes and the computation follows over

$$(\lambda n.E)\ \mathrm{n} \longrightarrow E\{\mathrm{n}/n\} \qquad \text{[App]}$$

$$\mathbf{R}\ S\ \lambda i.\lambda X.R\ 0 \longrightarrow S \qquad \text{[Zero]}$$

$$\mathbf{R}\ S\ \lambda i.\lambda X.R\ \mathrm{n}+1 \longrightarrow R\{\mathrm{n}/i\}\{\mathbf{R}\ S\ \lambda i.\lambda X.R\ \mathrm{n}/X\} \qquad \text{[Succ]}$$

$$\begin{array}{c}\bar{a}[\hat{\mathtt{p}}_0..\hat{\mathtt{p}}_n](y_0).R_0 \mid a[\hat{\mathtt{p}}_1](y_1).R_1 \mid ...a[\hat{\mathtt{p}}_n](y_n).R_n \\ \longrightarrow (\nu s)(R_0\{s[\hat{\mathtt{p}}_0]/y_0\} \mid ... \mid R_n\{s[\hat{\mathtt{p}}_n]/y_n\} \mid s:\emptyset)\end{array} \qquad \text{[Link]}$$

$$s[\hat{\mathtt{p}}]!\langle\hat{\mathtt{q}}, v\rangle; R \mid s:h \longrightarrow R \mid s:h\cdot(\hat{\mathtt{p}},\hat{\mathtt{q}},v) \qquad \text{[Send]}$$

$$s[\hat{\mathtt{p}}] \oplus \langle\hat{\mathtt{q}}, l\rangle; R \mid s:h \longrightarrow R \mid s:h\cdot(\hat{\mathtt{p}},\hat{\mathtt{q}},l) \qquad \text{[Label]}$$

$$s[\hat{\mathtt{p}}]?(\hat{\mathtt{q}}, x); R \mid s:(\hat{\mathtt{q}},\hat{\mathtt{p}},v)\cdot h \longrightarrow R\{v/x\} \mid s:h \qquad \text{[Recv]}$$

$$s[\hat{\mathtt{p}}]\&(\hat{\mathtt{q}}, \{l_i : R_i\}_{i\in I}) \mid s:(\hat{\mathtt{q}},\hat{\mathtt{p}},l_{i_0})\cdot h \longrightarrow R_{i_0} \mid s:h \quad (i_0 \in I) \qquad \text{[Branch]}$$

$$\mathsf{if}\ \mathbf{true}\ \mathsf{then}\ R\ \mathsf{else}\ S \longrightarrow R \qquad \mathsf{if}\ \mathbf{false}\ \mathsf{then}\ R\ \mathsf{else}\ S \longrightarrow S \qquad \text{[If-T, If-F]}$$

$$\mathsf{def}\ X(\tilde{y}\tilde{y}') = S\ \mathsf{in}\ (\mathbb{X}\langle\tilde{v}\tilde{s}[\hat{\mathtt{p}}]\rangle \mid R) \longrightarrow \mathsf{def}\ X(\tilde{y}\tilde{y}') = S\ \mathsf{in}\ (S\{\tilde{v}/\tilde{y}\}\{\tilde{s}[\hat{\mathtt{p}}]/\tilde{y}'\} \mid R) \qquad \text{[Def]}$$

$$R \longrightarrow R' \ \Rightarrow\ (\nu r)R \longrightarrow (\nu r)R' \qquad R \longrightarrow R' \ \Rightarrow\ R \mid S \longrightarrow R' \mid S \qquad \text{[Scop,Par]}$$

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

$$R \equiv R' \text{ and } R' \longrightarrow S' \text{ and } S \equiv S' \ \Rightarrow\ R \longrightarrow S \qquad \text{[Str]}$$

$$\mathtt{p}_i \downarrow \mathtt{p}'_i \ \Rightarrow\ \mathcal{R}[\mathtt{p}_0, ..., \mathtt{p}_i, ..., \mathtt{p}_n] \longrightarrow \mathcal{R}[\mathtt{p}_0, ..., \mathtt{p}'_i, ..., \mathtt{p}_n] \qquad \text{[ContextP]}$$

$$p \downarrow p' \ \Rightarrow\ \mathcal{R}[..., p] \longrightarrow \mathcal{R}[..., p'] \qquad \text{[ContextL]}$$

$$e_i \downarrow e'_i \ \Rightarrow\ \mathcal{E}[e_0, ..., e_i, ..., e_n] \longrightarrow \mathcal{E}[e_0, ..., e'_i, ..., e_n] \qquad \text{[ContextE]}$$

$$\mathtt{i}_i \downarrow \mathtt{i}'_i \ \Rightarrow\ \mathcal{N}[\mathtt{i}_0]...[\mathtt{i}_i]...[\mathtt{i}_n] \downarrow \mathcal{N}[\mathtt{i}_0]...[\mathtt{i}'_i]...[\mathtt{i}_n] \qquad \text{[Princ]}$$

Figure 3.7: CPS: Operational semantics.

value principals. The rule [Link] invokes a session between n peers by generating a session channel that substitutes it into the processes scope and a message queue. The identity of each principal within a session is represented by the label attached to the session channel. Below we give the definition of substitution:

Notation 3.4.3. Substitution is a function $\sigma = \{v_1, ..., v_n/x_1, ..., x_n\}$ and $\sigma = \{v_1, ..., v_n/n_1, ..., n_n\}$ that replaces respectively named variables and parameters by values such as $x_i\sigma = v_i$, $n_i\sigma = v_i$ for every $i \in [1..n]$ and $x\sigma = x$, $n\sigma = n$ if $x \notin \{x_1, ..., x_n\}$, $n \notin \{n_1, ..., n_n\}$.

Definition 3.4.4 (Substitution)**.** The expression $E\sigma$ obtained by applying σ to E is defined as follows:

$$\begin{array}{lcl}(\lambda n.\mathrm{E})\ \sigma & = & \lambda n.\mathrm{E}\sigma \\ (E\ \mathtt{t})\ \sigma & = & E\sigma\ \mathtt{t}\sigma\end{array}$$

$$
\begin{array}{lcl}
(\bar{u}[\mathtt{p}_0, \mathtt{p}_1, p](y).R)\ \sigma & = & \bar{u\sigma}[\mathtt{p}_0\sigma, \mathtt{p}_1\sigma, p\sigma](y).R\sigma \\
(u[\mathtt{p}](y).R)\ \sigma & = & u\sigma[\mathtt{p}\sigma](\tilde{y}).R\sigma \\
(k!\langle \mathtt{p}, e\rangle; R)\sigma & = & k\sigma!\langle \mathtt{p}\sigma, e\sigma\rangle; R\sigma \\
(k?\langle \mathtt{p}, x\rangle; R)\sigma & = & k\sigma?\langle \mathtt{p}\sigma, x\rangle; R\sigma \\
(k \oplus \langle \mathtt{p}, l\rangle; R)\sigma & = & k\sigma \oplus \langle \mathtt{p}\sigma, l\rangle; R\sigma \\
(k\&\langle \mathtt{p}, \{l_i : R_i\}_{i\in I}\rangle)\sigma & = & k\sigma\&\langle \mathtt{p}\sigma, \{l_i : R_i\sigma\}_{i\in I}\rangle \\
(\mathsf{def}\ X(\tilde{y}\tilde{y'}) = S\ \mathsf{in}\ R)\sigma & = & \mathsf{def}\ X(\tilde{y}\tilde{y'}) = S\sigma\ \mathsf{in}\ R\sigma \\
(X\langle \tilde{e}\tilde{k}\rangle)\sigma & = & X\langle \tilde{e}\sigma\tilde{k}\sigma\rangle \\
((\nu w)R)\sigma & = & (\nu w)R\sigma \\
(\mathbf{0})\ \sigma & = & \mathbf{0} \\
(R \mid S)\sigma & = & R\sigma \mid S\sigma \\
(\mathsf{if}\ e\ \mathsf{then}\ S\ \mathsf{else}\ R)\sigma & = & \mathsf{if}\ e\sigma\ \mathsf{then}\ S\sigma\ \mathsf{else}\ R\sigma \\
(\mathbf{R}\ S\ \lambda i.\lambda X.R)\sigma & = & \mathbf{R}\ S\sigma\ \lambda i.\lambda X.R\sigma \\
(X)\sigma & = & X \\
(R\ \mathtt{t})\ \sigma & = & R\sigma\ \mathtt{t}\sigma \\
(s : h)\sigma & = & s : h
\end{array}
$$

$$
\begin{array}{lcl}
(\mathtt{p}_1..\mathtt{p}_n)\sigma & = & \mathtt{p}_1\sigma..\mathtt{p}_n\sigma \\
(\mathbf{R}\ p\ \lambda i.\lambda X.p'\mathtt{t})\sigma & = & \mathbf{R}\ p\sigma\ \lambda i.\lambda X.p'\sigma\mathtt{t}\sigma \\
(e\ \mathtt{op}\ e')\sigma & = & e\sigma\ \mathtt{op}\ e'\sigma \\
(\mathtt{t}\ \mathrm{op}\ \mathrm{n})\ \sigma & = & \mathtt{t}\sigma\ \mathrm{op}\ \mathrm{n} \\
(\mathrm{n}^{\mathtt{t}})\ \sigma & = & \mathrm{n}^{\mathtt{t}\sigma}
\end{array}
$$

The [Send] and [Label] rules insert a message in the queue of the session. The result is the next term R from the sender process and a modified queue with the message append at the end. The message contains sender $\hat{\mathtt{p}}$, receiver $\hat{\mathtt{q}}$ and data: v or l.

The receiving rule [Recv] removes a value message of the same sender, as the one specified in the receiving construct, from the queue, and substitutes it in the process. The result is the next term of receiver R with values $\tilde{v}$ substituted in the scope and the modified queue without the message h. The [Branch] rule removes from the queue a label message of the same sender, as the ones specified in the branching construct. The result of the rule is the process following the label R_{i_0} and the modified queue without the message h.

Rule [IfT] and [IfF] action the evaluation of e; if e evaluates to **true** then rule [IfT] is applied, otherwise rule [IfF]. Rule [Def] invokes the process S identified by X with

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

$$(\nu w)R \mid \mathrm{S} \equiv (\nu w)(R \mid S) \quad \text{if } w \notin fn(\mathrm{S}) \cup fc(\mathrm{S})$$

$$(\nu w)\mathbf{0} \equiv \mathbf{0} \qquad \mathsf{def}\ D\ \mathsf{in}\ \mathbf{0} \equiv \mathbf{0}$$

$$\mathsf{def}\ D\ \mathsf{in}\ (\nu w)R \equiv (\nu w)\mathsf{def}\ D\ \mathsf{in}\ R \quad \text{if } w \notin fn(D) \cup fc(\mathrm{D})$$

$$(\mathsf{def}\ D\ \mathsf{in}\ \mathrm{R}) \mid \mathrm{S} \equiv \mathsf{def}\ D\ \mathsf{in}\ (\mathrm{R} \mid \mathrm{S}) \quad \text{if } \mathsf{dv}(D) \cap \mathrm{fpv}(S) = \emptyset$$

$$s : (\hat{\mathtt{q}}, \hat{\mathtt{p}}, z) \cdot (\hat{\mathtt{q}}', \hat{\mathtt{p}}', z') \cdot h \equiv s : (\hat{\mathtt{q}}', \hat{\mathtt{p}}', z') \cdot (\hat{\mathtt{q}}, \hat{\mathtt{p}}, z) \cdot h \qquad \text{if } \hat{\mathtt{p}} \neq \hat{\mathtt{p}}' \text{ or } \hat{\mathtt{q}} \neq \hat{\mathtt{q}}'$$

Figure 3.8: CPS: Structural congruence.

arguments $\tilde{v}$ in the context, resulting in the process S with parameters $\tilde{x}$ replaced by arguments $\tilde{v}$. The process R represents other calls to processes present in D, typically used to represent multiple recursive behaviour of different processes that are part of the same session. Rule [Scop] actions the reduction of the process inside the scope of the ν operator. Rule [Par] defines computation in a parallel composed process, reducing first the subprocesses. Similarly, rule [Defin] reduces first the subprocess R in the scope of the recursive process. Rule [Str] states that the reduction relation is defined on structural congruent terms; the reduction rule is identical to CSMS, with the exception that the congruence rules given in Figure 3.8 are extended with a rule on commutativity of messages in the queue, where $\mathsf{dv}(\{X_i(\tilde{x}) = S_i\}_{i \in I}) = \{X_i\}_{i \in I}$.

Definition 3.4.5. The set of free role variables of a general expression E, written $\mathrm{fpv}(E)$, is defined as follows:

$$\begin{aligned}
\mathrm{fpv}(\lambda n.E) &= \mathrm{fpv}(E) & \mathrm{fpv}(\mathsf{if}\ e\ \mathsf{then}\ S\ \mathsf{else}\ R) &= \mathrm{fpv}(S) \bigcup \mathrm{fpv}(R) \\
\mathrm{fpv}(E\ \mathtt{t}) &= \mathrm{fpv}(E) & \mathrm{fpv}(R \mid S) &= \mathrm{fpv}(R) \bigcup \mathrm{fpv}(S) \\
\mathrm{fpv}(\bar{u}[\mathtt{p}_0, \mathtt{p}_1, p](y).R) &= \mathrm{fpv}(R) & \mathrm{fpv}(u[\mathtt{p}](y).R) &= \mathrm{fpv}(R) \\
\mathrm{fpv}(k!\langle \mathtt{p}, e \rangle; R) &= \mathrm{fpv}(R) & \mathrm{fpv}((\nu w)R) &= \mathrm{fpv}(R) \\
\mathrm{fpv}(k?\langle \mathtt{p}, x \rangle; R) &= \mathrm{fpv}(R) & \mathrm{fpv}(X\langle \tilde{e}\tilde{k} \rangle) &= \{\mathrm{X}\} \\
\mathrm{fpv}(k \oplus \langle \mathtt{p}, l \rangle; R) &= \mathrm{fpv}(R) & \mathrm{fpv}(X) &= \{X\} \\
\mathrm{fpv}(k \& \langle \mathtt{p}, \{l_i : R_i\}_{i \in I} \rangle) &= \textstyle\bigcup_{i \in I} \mathrm{fpv}(R_i) & \mathrm{fpv}(R\ \mathtt{t}) &= \mathrm{fpv}(R) \\
\mathrm{fpv}(\mathbf{0}) &= \emptyset & \mathrm{fpv}(s : h) &= \emptyset
\end{aligned}$$

$$\mathrm{fpv}(\mathsf{def}\ X(\tilde{y}\tilde{y}') = S\ \mathsf{in}\ R) = \mathrm{fpv}(S) \bigcup \mathrm{fpv}(R) \setminus \{\mathrm{X}\}$$
$$\mathrm{fpv}(\mathbf{R}\ S\ \lambda i.\lambda X.R) = \mathrm{fpv}(S) \bigcup (\mathrm{fpv}(R) \setminus \{\mathrm{X}\})$$

Rule [ContextP] reduces the participants $\mathtt{p}_i$ inside the definition of processes $\mathcal{R}[\mathtt{p}_0, ..., \mathtt{p}_i, ..., \mathtt{p}_n]$ through the relation $\downarrow$. Reduction of principals $\mathcal{N}[\mathtt{i}_0]...[\mathtt{i}_i]...[\mathtt{i}_n]$ is defined by rule [Princ], reducing the mathematical expression present in every index expression $\mathtt{i}_i$. Rule

[ContextL] reduces the list of principals present in the prefix of processes requesting to initiate a session, where the reduction of parameterised list of principals is defined as:

$$\mathbf{R}\ p\ \lambda i.\lambda X.p'\ \ 0{\downarrow}\ p \qquad \text{[Zero]}$$

$$\mathbf{R}\ p\ \lambda i.\lambda X.p'\ \ \mathrm{n}{+}1{\downarrow}p'\{\mathrm{n}/i\}\{\mathbf{R}\ p\ \lambda i.\lambda X.p'\ \mathrm{n}/X\} \qquad \text{[Succ]}$$

Rule [ContextE] reduces the expressions, denoting messages inside the scope of sending actions, to values. Rule [Princ] reduces the index expressions inside the definition of principals.

Definition 3.4.6. The runtime terms $\mathcal{R}$ and $\mathcal{E}$ with the hole "[]" define the evaluation contexts of principals and expressions used by the last four reduction rules of the operational semantics to perform within the evaluation contexts below.

$\mathcal{R}[_, \ldots, _]$ ::=		Principals evaluation contexts
	$\bar{a}[_, ..., _](y).R$	Request
|	$a[_](y).R$	Accept
|	$s[\hat{\mathtt{p}}]!\langle _, e\rangle; R$	Send
|	$s[\hat{\mathtt{p}}] \oplus \langle _, l\rangle; R$	Selection
|	$s[\hat{\mathtt{p}}]?(_, x); R$	Receive
|	$s[\hat{\mathtt{p}}]\&(_, \{l_i : R_i\}_{i\in I})$	Branching
$\mathcal{E}[_, \ldots, _]$ ::=		Expressions evaluation contexts
|	_ `op` _	Expression
|	(R _)	Application
|	$c!\langle \mathtt{p}, _\rangle; R$	Send
|	(E _)	Application Expr.

Ring pattern The reduction steps of the Ring pattern for $n = 2$ described in Section 3.1 are given below where on the reduction relation $\longrightarrow$ is attached the name of the operational semantics rule applied. The first step invokes the main program by replacing the parameter n with the argument 2 (rule [App]) and reduces to a natural number: 1) the participant expression in *Last*, namely in the receive action (rule [ContextP]), and the argument applied to the **R** expression (rule [ContextE]). The second step reduces the parameterised list of principals in *Starter*. At this point, *Starter* and *Last* are instantiated to running processes. In the next two steps, *Middle* is instantiated by replacing the index i with 0 and the process variable X is replaced by the processes created in the first two

steps. Subsequently, a session of Ring of length three starts between the running processes, including the creation of a session channel s and a queue with the same identifier. The subsequent steps are intuitive: appending messages in the queue in case of "send" and reciprocally, removing and substituting them in processes scope in case of "receive". The message structure contains the sender, receiver and the data sent.

$$\mathit{Ring}\ 2 \rightarrow_{[App],[ContextE],[ContextP]}$$

$$\mathbf{R}\ \bar{a}[\mathtt{W}[0], \mathtt{W}[1], \mathtt{W}\{2/n\}](y).y!\langle \mathtt{W}[1], v\rangle; y?(\mathtt{W}[2], z); P \mid a[\mathtt{W}[2]](y).y?(\mathtt{W}[1], z); y!\langle \mathtt{W}[0], z\rangle; Q$$
$$\lambda i.\lambda X.(\mathit{Middle(i)} \mid X)\ 1 \rightarrow_{[ContextL]}$$

$$\mathbf{R}\ \bar{a}[\mathtt{W}[0], \mathtt{W}[1], \mathtt{W}[2]](y).y!\langle \mathtt{W}[1], v\rangle; y?(\mathtt{W}[2], z); P \mid a[\mathtt{W}[2]](y).y?(\mathtt{W}[1], z); y!\langle \mathtt{W}[0], z\rangle; Q$$
$$\lambda i.\lambda X.(\mathit{Middle(i)} \mid X)\ 1 \rightarrow_{[Succ],[ContextP]}$$

$$a[\mathtt{W}[1]](y).y?(\mathtt{W}[0], z); y!\langle \mathtt{W}[2], z\rangle;$$
$$\mid X\{\mathbf{R}$$
$$\bar{a}[\mathtt{W}[0], \mathtt{W}[1], \mathtt{W}[2]](y).y!\langle \mathtt{W}[1], v\rangle; y?(\mathtt{W}[2], z); P \mid a[\mathtt{W}[2]](y).y?(\mathtt{W}[1], z); y!\langle \mathtt{W}[0], z\rangle; Q$$
$$\lambda i.\lambda X.(\mathit{Middle(i)} \mid X)\ 0/X\} \rightarrow_{[Zero]}$$

$$a[\mathtt{W}[1]](y).y?(\mathtt{W}[0], z); y!\langle \mathtt{W}[2], z\rangle; \mid \bar{a}[\mathtt{W}[0], \mathtt{W}[1], \mathtt{W}[2]](y).y!\langle \mathtt{W}[1], v\rangle; y?(\mathtt{W}[2], z); P$$
$$\mid a[\mathtt{W}[2]](y).y?(\mathtt{W}[1], z); y!\langle \mathtt{W}[0], z\rangle; Q \rightarrow_{[Link]}$$

$$(\nu s)(s[\mathtt{W}[1]]?(\mathtt{W}[0], z); s[\mathtt{W}[1]]!\langle \mathtt{W}[2], z\rangle; \mid s[\mathtt{W}[0]]!\langle \mathtt{W}[1], v\rangle; s[\mathtt{W}[0]]?(\mathtt{W}[2], z); P$$
$$\mid s[\mathtt{W}[2]]?(\mathtt{W}[1], z); s[\mathtt{W}[2]]!\langle \mathtt{W}[0], z\rangle; Q \mid s : \emptyset) \rightarrow_{[Scop],[Par],[Send]}$$

$$(\nu s)(s[\mathtt{W}[1]]?(\mathtt{W}[0], z); s[\mathtt{W}[1]]!\langle \mathtt{W}[2], z\rangle; \mid s[\mathtt{W}[0]]?(\mathtt{W}[2], z); P$$
$$\mid s[\mathtt{W}[2]]?(\mathtt{W}[1], z); s[\mathtt{W}[2]]!\langle \mathtt{W}[0], z\rangle; Q \mid s : \langle \mathtt{W}[0], \mathtt{W}[1], v\rangle) \rightarrow_{[Scop],[Par],[Recv]}$$

$$(\nu s)(s[\mathtt{W}[1]]!\langle \mathtt{W}[2], v\rangle; \mid s[\mathtt{W}[0]]?(\mathtt{W}[2], z); P \mid s[\mathtt{W}[2]]?(\mathtt{W}[1], z); s[\mathtt{W}[2]]!\langle \mathtt{W}[0], z\rangle; Q \mid s : \emptyset)$$
$$\rightarrow_{[Scop],[Par],[Send]}$$

$$(\nu s)(s[\mathtt{W}[0]]?(\mathtt{W}[2], z); P \mid s[\mathtt{W}[2]]?(\mathtt{W}[1], z); s[\mathtt{W}[2]]!\langle \mathtt{W}[0], z\rangle; Q \mid s : \langle \mathtt{W}[1], \mathtt{W}[2], v\rangle)$$
$$\rightarrow_{[Scop],[Par],[Recv]}$$

$$\frac{C \vdash \mathtt{p}, \mathtt{p}' \quad \Theta; C \vdash G' \blacktriangleright \mathsf{Type} \quad \emptyset; C \vdash \tilde{U} \blacktriangleright \mathsf{Type}}{\Theta; C \vdash \mathtt{p} \to \mathtt{p}' : \langle \tilde{U} \rangle . G' \blacktriangleright \mathsf{Type}} \lfloor \text{KIO} \rfloor \qquad \frac{C \vdash \mathtt{p}, \mathtt{p}' \quad \forall i \in I,\ \Theta; C \vdash G_i \blacktriangleright \mathsf{Type}}{\Theta; C \vdash \mathtt{p} \to \mathtt{p}' : \{l_i : G_i\}_{i \in I} \blacktriangleright \mathsf{Type}} \lfloor \text{KBRA} \rfloor$$

$$\frac{\Theta; C \vdash G \blacktriangleright \mathsf{Type} \quad \mathbf{x} \in \mathsf{fprtv}(G') \quad \Theta, \mathbf{x} : \mathsf{Type}; C, i : \mathtt{I} \vdash G' \blacktriangleright \mathsf{Type}}{\Theta; C \vdash \mathbf{R}\ G\ \lambda i . \lambda \mathbf{x} . G' \blacktriangleright \Pi i : \mathtt{I} . \mathsf{Type}} \lfloor \text{KPR} \rfloor \qquad \frac{C \vdash \mathtt{t} \quad \Theta; C \vdash G \blacktriangleright \Pi i : \{i | i \in \mathsf{nat}, 0 \leq i \leq \mathtt{t}-1\} . \mathsf{Type}}{\Theta; C \vdash G\ \mathtt{t} \blacktriangleright \mathsf{Type}} \lfloor \text{KAPP} \rfloor$$

$$\frac{}{\Theta, \mathbf{t} : \mathsf{Type}; C \vdash \mathbf{t} \blacktriangleright \mathsf{Type}} \lfloor \text{KRVAR} \rfloor \qquad \frac{}{\Theta, \mathbf{x} : \mathsf{Type}; C \vdash \mathbf{x} \blacktriangleright \mathsf{Type}} \lfloor \text{KPRVAR} \rfloor$$

$$\frac{\Theta, \mathbf{t} : \mathsf{Type}; C \vdash G \blacktriangleright \mathsf{Type}}{\Theta; C \vdash \mu \mathbf{t} . G \blacktriangleright \mathsf{Type}} \lfloor \text{KREC} \rfloor \qquad \frac{}{\Theta; C \vdash \mathsf{end} \blacktriangleright \mathsf{Type}} \lfloor \text{KEND} \rfloor$$

Figure 3.9: CPS: Well-formedness of global types.

$$(\nu s)(s[\mathtt{W}[0]]?(\mathtt{W}[2], z); P \mid s[\mathtt{W}[2]]!\langle \mathtt{W}[0], v \rangle; Q \mid s : \emptyset) \rightarrow_{[Scop],[Par],[Send]}$$

$$(\nu s)(s[\mathtt{W}[0]]?(\mathtt{W}[2], z); P \mid Q \mid s : \langle \mathtt{W}[2], \mathtt{W}[0], v \rangle) \rightarrow_{[Scop],[Par],[Recv]}$$

$$(\nu s)(P\{v/z\} \mid Q \mid s : \emptyset)$$

3.4.3 Well-formedness of Global Types

Global types in CPS are constructed by the message, branching, recursion, primitive recursion and application constructs. Primitive recursion forms a new kind of global type—product kind. Well-formedness rules, presented as a kinding system, shown in Figure 3.9, ensure that in a global type the indexed principals are well-formed and parametric expressions are applied only to product global types. They are of the form $\Theta; C \vdash G \blacktriangleright \kappa$, read, "In the variable typing Θ and parameter-index typing C, global type G has kind κ".

Definition 3.4.7. Below we give the formal definition of contexts and types:

$$\Theta ::= \emptyset \mid \mathbf{x} : \mathsf{Type}, \Theta \mid \mathbf{t} : \mathsf{Type}, \Theta \qquad C ::= \emptyset \mid n : \mathtt{T}, C \mid i : \mathtt{I}, C \qquad \kappa ::= \mathsf{Type} \mid \Pi i : \mathtt{I} . \mathsf{Type}$$
$$\mathtt{T} ::= \{n \mid n \in \mathsf{nat}, n \geq \mathrm{n}\} \qquad \mathtt{I} ::= \{i \mid i \in \mathsf{nat}, 0 \leq i \leq \mathtt{t}\}$$

The typing context Θ is a sequence of primitive recursive variables and recursive variables with their kind Type. The typing context C is a sequence of indexes and parameters (recalling from Section 3.4.1: both are special variables) with their type $\mathtt{I}$ and $\mathtt{T}$. The type $\mathtt{I}$ is a finite set of natural numbers ranging from 0 to a parametric expression that ranges over natural numbers and the type $\mathtt{T}$ is an infinite set of natural numbers that is defined only by a lower bound n. The kind κ is a Type, representing base global types, and $\Pi i : \mathtt{I}.\mathsf{Type}$, representing a product global type.

The $\lfloor$KIO$\rfloor$ rule ensures that the principals of a causality are well-formed, the message type is kinded under an empty context, and that the inductive part is Type kinded. The context of well-formedness is empty for the same reason as explained in CSMS (see Section 2.6.3). A well-formed principal is either a participant $\mathcal{N}$ ($\mathtt{Alice}$, $\mathtt{Bob}$, ...) or an indexed principal where the index expression is well-formed.

Definition 3.4.8 (Well-formedness of principals)**.** A principal $\mathtt{p}$ is well-formed, written $C \vdash \mathtt{p}$ and read, "In the parameter-index typing C, principal $\mathtt{p}$ is well-formed" if

- $\mathtt{p} = \mathcal{N}$ then $C \vdash \mathcal{N}$.
- $\mathtt{p} = \mathtt{p}'[\mathtt{i}]$ and $C \vdash \mathtt{i} \quad C \vdash \mathtt{p}'$ then $C \vdash \mathtt{p}'[\mathtt{i}]$.

A well-formed index expression is an index present in the typing context and an index expression ranging over natural numbers defined on a safe form of subtraction. Subtraction is defined only between an index expression and a parametric expression where the maximum value of the former is less than the latter.

Definition 3.4.9 (Well-formedness of index expressions)**.** An index expression $\mathtt{i}$ is well-formed, written $C \vdash \mathtt{i}$ and read, "In the parameter-index typing C, index expression $\mathtt{i}$ is well-formed" if

- $\mathtt{i} = i$ and $i \in dom(C)$ then $C \vdash i$.
- $\mathtt{i} = \mathrm{n} * \mathtt{i}'$ and $C \vdash \mathtt{i}'$ then $C \vdash \mathrm{n} * \mathtt{i}'$.
- $\mathtt{i} = \mathtt{t} + \mathtt{i}'$ and $C \vdash \mathtt{t} \quad C \vdash \mathtt{i}'$ then $C \vdash \mathtt{t} + \mathtt{i}'$.
- $\mathtt{i} = \mathtt{t} - \mathtt{i}'$ and $C \vdash \mathtt{t} \geq \mathtt{t}'$, where $C \vdash \max(\mathtt{i}') = \mathtt{t}'$, then $C \vdash \mathtt{t} - \mathtt{i}'$.

Definition 3.4.10. The maximum value of an index expression, written $C \vdash \max(\mathtt{i}) = \mathtt{t}$ and read "In the parameter-index typing C, index expression $\mathtt{i}$ has maximum value the parametric expression $\mathtt{t}$", is defined inductively on every construct of index expressions as:

$$\frac{}{C \vdash \mathrm{n} \geq 0} \quad \frac{C \vdash n : \{n | n \geq \mathrm{n}\}}{C \vdash n \geq 0} \quad \frac{C \vdash \mathtt{t} \ \geq 0}{C \vdash \mathtt{t} * \mathrm{n} \geq 0}$$

$$\frac{C \vdash \mathtt{t} \geq \mathrm{n}}{C \vdash \mathtt{t} - \mathrm{n} \geq 0} \quad \frac{C \vdash \mathtt{t} \geq 0}{C \vdash \mathtt{t} + \mathrm{n} \geq 0} \quad \frac{C \vdash \mathtt{t} \geq 0}{C \vdash n^{\mathtt{t}} \geq 0}$$

$$\frac{\mathrm{n} \geq \mathrm{n}'}{C \vdash \mathrm{n} \geq \mathrm{n}'} \quad \frac{C \vdash n : \{n | n \geq \mathrm{n}'\} \quad \mathrm{n}' \geq \mathrm{n}}{C \vdash n \geq \mathrm{n}} \quad \frac{C \vdash \mathtt{t} \geq \mathrm{n}'/\ \mathrm{n} \quad \mathrm{n}' \% \mathrm{n} = 0}{C \vdash \mathtt{t} * \mathrm{n} \geq \mathrm{n}'}$$

$$\frac{C \vdash \mathtt{t} \geq \mathrm{n}' - \mathrm{n}}{C \vdash \mathtt{t} + \mathrm{n} \geq \mathrm{n}'} \quad \frac{C \vdash \mathtt{t} \geq \mathrm{n}' + \mathrm{n}}{C \vdash \mathtt{t} - \mathrm{n} \geq \mathrm{n}'} \quad \frac{C \vdash \mathtt{t} \geq \lg_{\mathrm{n}} \mathrm{n}' \quad \mathrm{n}' = \mathrm{n}^{\mathrm{n}''}}{C \vdash \mathrm{n}^{\mathtt{t}} \geq \mathrm{n}'}$$

$$\frac{C \vdash \mathtt{t}' - \mathtt{t} \geq 0}{C \vdash \mathtt{t} \leq \mathtt{t}'}$$

Figure 3.10: CPS: Inequality between parameters.

- $C \vdash \max(\mathtt{t}) = \mathtt{t}$
- $C \vdash \max(i) = \max(C(i))$
- $C \vdash \max(\mathrm{n} * \mathtt{i}) = \mathrm{n} * C \vdash \max(\mathtt{i})$
- $C \vdash \max(\mathtt{t} \pm \mathtt{i}) = \mathtt{t} \pm C \vdash \max(\mathtt{i})$

Definition 3.4.11. The maximum value of an index range is defined as:

$$\mathsf{max}(\{i \mid i \in \mathsf{nat}, 0 \leq i \leq \mathtt{t}\}) = \mathtt{t}$$

Figure 3.10 defines the inequality between parametric expressions, transforming them into inequalities of the form "greater than a natural number, including zero" and then checking whether the minimum value of the parametric expression (natural number) is greater than the natural number.

Proposition 3.4.12 (Decidability of Inequality of Parameters)**.** The relation $C \vdash \mathtt{t} \geq \mathrm{n}$ is decidable.

Proof. The statement that "$C \vdash \mathtt{t} \geq \mathrm{n}$ is decidable" follows directly by induction hypothesis on every rule of the system in Figure 3.10. We present the most interesting cases. The other cases are symmetrical.

Case: $C \vdash \mathtt{t} * \mathrm{n} \geq 0$ is decidable since $C \vdash \mathtt{t} \geq \mathrm{n}$ is decidable by induction hypothesis.
Case: $C \vdash \mathrm{n} \geq \mathrm{n}'$ is decidable since $\mathrm{n} \geq \mathrm{n}'$ is decidable. □

Proposition 3.4.13 (Decidability of Index expressions Well-formedness)**.** The relation $C \vdash \mathtt{i}$ is decidable.

Proof. The statement that "$C \vdash \mathtt{i}$ is decidable" follows directly by induction hypothesis on every rule of the system in Definition 3.4.9. We present the most interesting cases. The other cases are symmetrical.

Case: $C \vdash \mathtt{i} * \mathrm{n}$ is decidable since $C \vdash \mathtt{i}$ is decidable by induction hypothesis.
Case: $C \vdash \mathtt{t} - \mathtt{i}$ is decidable since $C \vdash \mathtt{t} \geq \mathtt{t}'$ (from $C \vdash \mathtt{t} - \mathtt{t}' \geq 0$) is decidable by Proposition 3.4.12. □

A well-formed parametric expression is a natural number, a parameter present in the context, and a parametric expression ranging over natural number defined on a safe form of subtraction. Subtraction is defined only between a parametric expression and a natural where the former is greater than the latter.

Definition 3.4.14 (Well-formedness of parametric expressions)**.** A parametric expression $\mathtt{t}$ is well-formed, written $C \vdash \mathtt{t}$ and read, "In the parameter-index typing C, parametric expression $\mathtt{t}$ is well-formed" if

- $\mathtt{t} = \mathrm{n}$ then $C \vdash \mathrm{n}$.
- $\mathtt{t} = n$ and $n \in dom(C)$ then $C \vdash n$.
- $\mathtt{t} = \mathtt{t}' * \mathrm{n}$ and $C \vdash \mathtt{t}'$ then $C \vdash \mathtt{t}' * \mathrm{n}$.
- $\mathtt{t} = \mathtt{t}' + \mathrm{n}$ and $C \vdash \mathtt{t}'$ then $C \vdash \mathtt{t}' + \mathrm{n}$.
- $\mathtt{t} = \mathrm{n}^{\mathtt{t}'}$ and $C \vdash \mathtt{t}'$ then $C \vdash \mathrm{n}^{\mathtt{t}'}$.
- $\mathtt{t} = \mathtt{t}' - \mathrm{n}$ and $C \vdash \mathtt{t}' \geq \mathrm{n}$ then $C \vdash \mathtt{t}' - \mathrm{n}$.

Proposition 3.4.15 (Decidability of Parametric expressions Well-formedness)**.** The relation $C \vdash \mathtt{t}$ is decidable.

Proof. The statement that "$C \vdash \mathtt{t}$ is decidable" follows directly by induction hypothesis on every rule of the system in Definition 3.4.14. We present the most interesting cases. The other cases are symmetrical.

Case: $C \vdash \mathtt{t} * \mathrm{n}$ is decidable since $C \vdash \mathtt{t}$ is decidable by induction hypothesis.
Case: $C \vdash \mathtt{t} - \mathrm{n}$ is decidable since $C \vdash \mathtt{t} \geq \mathrm{n}$ is decidable by Proposition 3.4.12. □

$$\frac{}{\Theta; C \vdash \mathsf{bool} \blacktriangleright \mathsf{Type}} \lfloor \textsc{KBool} \rfloor \quad \frac{}{\Theta; C \vdash \mathsf{nat} \blacktriangleright \mathsf{Type}} \lfloor \textsc{KNat} \rfloor \quad \frac{\Theta; C \vdash G \blacktriangleright \mathsf{Type}}{\Theta; C \vdash \langle G \rangle \blacktriangleright \mathsf{Type}} \lfloor \textsc{KMar} \rfloor$$

$$\frac{C \vdash \mathtt{p} \quad \Theta; C \vdash U \blacktriangleright \mathsf{Type} \quad \mathsf{fprtv}(U) = \emptyset \quad \Theta; C \vdash T \blacktriangleright \mathsf{Type}}{\Theta; C \vdash !\langle \mathtt{p}, U \rangle; T \blacktriangleright \mathsf{Type}} \lfloor \textsc{KROut} \rfloor$$

$$\frac{C \vdash \mathtt{p} \quad \Theta; C \vdash U \blacktriangleright \mathsf{Type} \quad \mathsf{fprtv}(U) = \emptyset \quad \Theta; C \vdash T \blacktriangleright \mathsf{Type}}{\Theta; C \vdash ?\langle \mathtt{p}, U \rangle; T \blacktriangleright \mathsf{Type}} \lfloor \textsc{KRIn} \rfloor$$

$$\frac{C \vdash \mathtt{p} \quad \forall i \in I. \Theta; C \vdash T_i \blacktriangleright \mathsf{Type}}{\Theta; C \vdash \oplus\langle \mathtt{p}, \{l_i : T_i\}_{i \in I} \rangle \blacktriangleright \mathsf{Type}} \lfloor \textsc{KRSel} \rfloor \quad \frac{C \vdash \mathtt{p} \quad \forall i \in I. \Theta; C \vdash T_i \blacktriangleright \mathsf{Type}}{\Theta; C \vdash \&\langle \mathtt{p}, \{l_i : T_i\}_{i \in I} \rangle \blacktriangleright \mathsf{Type}} \lfloor \textsc{KRBra} \rfloor$$

$$\frac{\Theta, \mathbf{t} : \mathsf{Type}; C \vdash T \blacktriangleright \mathsf{Type}}{\Theta; C \vdash \mu \mathbf{t}.T \blacktriangleright \mathsf{Type}} \lfloor \textsc{KRRec} \rfloor \quad \frac{}{\Theta, \mathbf{t} : \mathsf{Type}; C \vdash \mathbf{t} \blacktriangleright \mathsf{Type}} \lfloor \textsc{KRVar} \rfloor$$

$$\frac{\Theta; C \vdash T \blacktriangleright \mathsf{Type} \quad \Theta, \mathbf{x} : \mathsf{Type}; C, i : \mathtt{I} \vdash T' \blacktriangleright \mathsf{Type}}{\Theta; C \vdash \mathbf{R}\ T\ \lambda i. \lambda \mathbf{x}. T' \blacktriangleright \Pi i : \mathtt{I}. \mathsf{Type}} \lfloor \textsc{KRPRec} \rfloor$$

$$\frac{}{\Theta, \mathbf{t} : \mathsf{Type}; C \vdash \mathbf{x} \blacktriangleright \mathsf{Type}} \lfloor \textsc{KRTVar} \rfloor \quad \frac{}{\Theta; C \vdash \mathsf{end} \blacktriangleright \mathsf{Type}} \lfloor \textsc{KREnd} \rfloor$$

$$\frac{C \vdash \mathtt{t} \quad \Theta; C \vdash T \blacktriangleright \Pi i : \{i | i \in \mathsf{nat}, 0 \leq i \leq \mathtt{t} - 1\}. \mathsf{Type}}{\Theta; C \vdash T\ \mathtt{t} \blacktriangleright \mathsf{Type}} \lfloor \textsc{KApp} \rfloor$$

Figure 3.11: CPS: Well-formedness of value and role types.

Proposition 3.4.16 (Decidability of Principal Well-formedness)**.** The relation $C \vdash \mathtt{p}$ is decidable.

Proof. The statement that "$C \vdash \mathtt{p}$ is decidable" follows directly by induction hypothesis on every rule of the system in Definition 3.4.8.

Case: $C \vdash \mathcal{N}$ is decidable by rule.
Case: $C \vdash \mathtt{p}[\mathtt{i}]$ is decidable since $C \vdash \mathtt{p}$ is decidable by induction hypothesis and $C \vdash \mathtt{i}$ is decidable by Proposition 3.4.13. □

The $\lfloor$KBra$\rfloor$ rule checks that the principals of label causality are well-formed and that the inductive parts are Type kinded.

The $\lfloor$KPR$\rfloor$ rule ensures that the inductive parts are Type kinded, returning the kind of a product global type. The rule also ensures that the primitive recursive variable appears

in the scope of the recursive term to enforce recursion in the global type; e.g. the following global type:

$$\mathbf{R}\ \mathtt{W}[n] \to \mathtt{W}[0] : \langle U \rangle.\mathsf{end}\ \lambda j.\lambda \mathbf{y}.\mathtt{W}[n-j-1] \to \mathtt{W}[n-j] : \langle U \rangle.\mathsf{end}\ \mathrm{n}$$

does not define a ring of length n but a single causality, $\mathtt{W}[0] \to \mathtt{W}[1] : \langle U \rangle.\mathsf{end}$. This is a convention used by the type system to check the number of participants in global types and processes as we shall see in Section 3.4.6 when typing the Ring.

Definition 3.4.17. The set of free primitive recursion variables in global types G, denoted $\mathsf{fprtv}(G)$, is defined as follows:

$$\begin{gathered}
\mathsf{fprtv}(\mathtt{p} \to \mathtt{p}' : \langle U \rangle.G) = \mathsf{fprtv}(G) \cup \mathsf{fprtv}(U) \\
\mathsf{fprtv}(\mathtt{p} \to \mathtt{p}' : \{l_i : G_i\}_{i \in I}) = \textstyle\bigcup_{i \in I} \mathsf{fprtv}(G_i) \\
\mathsf{fprtv}(\mu \mathbf{t}.G) = \mathsf{fprtv}(G) \quad \mathsf{fprtv}(\mathbf{t}) = \emptyset \\
\mathsf{fprtv}(\mathbf{R}\ G\ \lambda i.\lambda \mathbf{x}.G') = \mathsf{fprtv}(G) \cup \mathsf{fprtv}(G') \setminus \{\mathbf{x}\} \\
\mathsf{fprtv}(G\ \mathtt{t}) = \mathsf{fprtv}(G) \quad \mathsf{fprtv}(\mathbf{x}) = \{\mathbf{x}\} \quad \mathsf{fprtv}(\mathsf{end}) = \emptyset \\
\mathsf{fprtv}(\mathsf{bool}) = \emptyset \quad \mathsf{fprtv}(\langle G \rangle) = \mathsf{fprtv}(G) \\
\mathsf{fprtv}(!\langle \mathtt{p}, U \rangle; T) = \mathsf{fprtv}(T) \cup \mathsf{fprtv}(U) \\
\mathsf{fprtv}(?\langle \mathtt{p}, U \rangle; T) = \mathsf{fprtv}(T) \cup \mathsf{fprtv}(U) \\
\mathsf{fprtv}(\oplus \langle \mathtt{p}, \{l_i : T_i\}_{i \in I} \rangle) = \textstyle\bigcup_{i \in I} \mathsf{fprtv}(T_i) \\
\mathsf{fprtv}(\& \langle \mathtt{p}, \{l_i : T_i\}_{i \in I} \rangle) = \textstyle\bigcup_{i \in I} \mathsf{fprtv}(T_i) \\
\mathsf{fprtv}(\mu \mathbf{t}.T) = \mathsf{fprtv}(T) \quad \mathsf{fprtv}(\mathbf{t}) = \emptyset \\
\mathsf{fprtv}(\mathbf{R}\ T\ \lambda i.\lambda \mathbf{x}.T') = \mathsf{fprtv}(T) \cup \mathsf{fprtv}(T') \setminus \{\mathbf{x}\} \\
\mathsf{fprtv}(T\ \mathtt{i}) = \mathsf{fprtv}(T) \quad \mathsf{fprtv}(\mathbf{x}) = \{\mathbf{x}\} \quad \mathsf{fprtv}(\mathsf{end}) = \emptyset
\end{gathered}$$

The $\lfloor$KApp$\rfloor$ rule checks that the argument is applied to a product global type, ranges over natural numbers, and that it is a successor of the biggest index value, returning a Type kind.

Other rules ensure that the inductive global types are Type kinded and look up for type variables in the context Θ.

Figure 3.11 gives the auxiliary definitions for kinding of value and role types used in the global kinding rules. The primitive types bool and nat are Type kinded and the shared name global type is Type kinded. The role type kinding rules ensure that in a role type the indexed principals are well-formed and parametric expressions are applied only to product global types; the judgment reads the same as global type kinding judgments.

Proposition 3.4.18 (Decidability of Value Type Kinding)**.** The relation $\Theta; C \vdash U \blacktriangleright \kappa$ is decidable.

Proof. Follows by induction hypothesis of kinding rules in Figures 3.11.

$\Theta; C \vdash \langle G \rangle \blacktriangleright \mathsf{Type}$	By assumption
$\Theta; C \vdash G \blacktriangleright \mathsf{Type}$	By inversion
$\Theta; C \vdash G \blacktriangleright \mathsf{Type}$ is decidable	By Proposition 3.4.20
$\Theta; C \vdash \langle G \rangle \blacktriangleright \mathsf{Type}$ is decidable	By rule

Other cases are straightforward. □

Proposition 3.4.19 (Decidability of Role Type Kinding)**.** The relation $\Theta; C \vdash T \blacktriangleright \kappa$ is decidable.

Proof. Follows by induction hypothesis of kinding rules in Figures 3.11.

$\Theta; C \vdash !\langle \mathtt{p}, U \rangle; T \blacktriangleright \mathsf{Type}$	By assumption
$C \vdash \mathtt{p} \quad \Theta; C \vdash U \blacktriangleright \mathsf{Type} \quad \mathsf{fprtv}(U) = \emptyset \quad \Theta; C \vdash T \blacktriangleright \mathsf{Type}$	By inversion
$C \vdash \mathtt{p}$ is decidable	By Proposition 3.4.16
$\Theta; C \vdash U \blacktriangleright \mathsf{Type}$ is decidable	By Proposition 3.4.18
$\Theta; C \vdash T \blacktriangleright \mathsf{Type}$ is decidable	By i.h.
$\Theta; C \vdash !\langle \mathtt{p}, U \rangle; T \blacktriangleright \mathsf{Type}$ is decidable	By rule

$\Theta; C \vdash ?\langle \mathtt{p}, U \rangle; T \blacktriangleright \mathsf{Type}$	By assumption
$C \vdash \mathtt{p} \quad \Theta; C \vdash U \blacktriangleright \mathsf{Type} \quad \mathsf{fprtv}(U) = \emptyset \quad \Theta; C \vdash T \blacktriangleright \mathsf{Type}$	By inversion
$C \vdash \mathtt{p}$ is decidable	By Proposition 3.4.16
$\Theta; C \vdash U \blacktriangleright \mathsf{Type}$ is decidable	By Proposition 3.4.18
$\Theta; C \vdash T \blacktriangleright \mathsf{Type}$ is decidable	By i.h.
$\Theta; C \vdash ?\langle \mathtt{p}, U \rangle; T \blacktriangleright \mathsf{Type}$ is decidable	By rule

$\Theta; C \vdash \oplus\langle \mathtt{p}, \{l_i : T_i\}_{i \in I} \rangle \blacktriangleright \mathsf{Type}$	By assumption
$C \vdash \mathtt{p} \quad \forall i \in I. \Theta; C \vdash T_i \blacktriangleright \mathsf{Type}$	By inversion
$C \vdash \mathtt{p}$ is decidable	By Proposition 3.4.16
$\forall i \in I. \Theta; C \vdash T_i \blacktriangleright \mathsf{Type}$ is decidable	By i.h.

$\Theta; C \vdash \oplus\langle \mathtt{p}, \{l_i : T_i\}_{i\in I}\rangle \blacktriangleright \mathsf{Type}$ is decidable — By rule

$\Theta; C \vdash \&\langle \mathtt{p}, \{l_i : T_i\}_{i\in I}\rangle \blacktriangleright \mathsf{Type}$ — By assumption
$C \vdash \mathtt{p} \quad \forall i \in I. \Theta; C \vdash T_i \blacktriangleright \mathsf{Type}$ — By inversion
$C \vdash \mathtt{p}$ is decidable — By Proposition 3.4.16
$\forall i \in I. \Theta; C \vdash T_i \blacktriangleright \mathsf{Type}$ is decidable — By i.h.
$\Theta; C \vdash \&\langle \mathtt{p}, \{l_i : T_i\}_{i\in I}\rangle \blacktriangleright \mathsf{Type}$ is decidable — By rule

$\Theta; C \vdash \mathbf{R}\ T\ \lambda i.\lambda \mathbf{x}.T' \blacktriangleright \Pi i : \mathtt{I}.\mathsf{Type}$ — By assumption
$\Theta; C \vdash T \blacktriangleright \mathsf{Type} \quad \Theta, \mathbf{x} : \mathsf{Type}; C, i : \mathtt{I} \vdash T' \blacktriangleright \mathsf{Type}$ — By inversion
$\Theta; C \vdash T \blacktriangleright \mathsf{Type}$ is decidable — By i.h.
$\Theta, \mathbf{x} : \mathsf{Type}; C, i : \mathtt{I} \vdash T' \blacktriangleright \mathsf{Type}$ is decidable — By i.h.
$\Theta; C \vdash \mathbf{R}\ T\ \lambda i.\lambda \mathbf{x}.T' \blacktriangleright \Pi i : \mathtt{I}.\mathsf{Type}$ is decidable — By rule

$\Theta; C \vdash T\ \mathtt{t} \blacktriangleright \mathsf{Type}$ — By assumption
$C \vdash \mathtt{t} \quad \Theta; C \vdash T \blacktriangleright \Pi i : \{i | i \in \mathsf{nat}, 0 \leq i \leq \mathtt{t} - 1\}.\mathsf{Type}$ — By inversion
$C \vdash \mathtt{t}$ is decidable — By Proposition 3.4.15
$\Theta; C \vdash T \blacktriangleright \Pi i : \{i | i \in \mathsf{nat}, 0 \leq i \leq \mathtt{t} - 1\}.\mathsf{Type}$ is decidable — By i.h.
$\Theta; C \vdash T\ \mathtt{t} \blacktriangleright \mathsf{Type}$ is decidable — By rule

Other cases are trivial by induction. □

Proposition 3.4.20 (Decidability of Kinding)**.** The relation $\Theta; C \vdash G \blacktriangleright \kappa$ is decidable.

Proof. Follows by induction hypothesis of kinding rules in Figures 3.9.

$\Theta; C \vdash \mathtt{p} \to \mathtt{p}' : \langle \tilde{U} \rangle.G' \blacktriangleright \mathsf{Type}$ — By assumption
$C \vdash \mathtt{p}, \mathtt{p}' \quad \Theta; C \vdash G' \blacktriangleright \mathsf{Type} \quad \emptyset; C \vdash \tilde{U} \blacktriangleright \mathsf{Type}$ — By inversion
$C \vdash \mathtt{p}, \mathtt{p}'$ is decidable — By Proposition 3.4.16
$\Theta; C \vdash G' \blacktriangleright \mathsf{Type}$ is decidable — By i.h.
$\emptyset; C \vdash \tilde{U} \blacktriangleright \mathsf{Type}$ is decidable — By Proposition 3.4.19
$\Theta; C \vdash \mathtt{p} \to \mathtt{p}' : \langle \tilde{U} \rangle.G' \blacktriangleright \mathsf{Type}$ is decidable — By rule

$T ::= !\langle \mathtt{p}, \tilde{U} \rangle; T$	Output	$\mid \mathbf{R}\ T\ \lambda i.\lambda \mathbf{x}.T'$	Primitive Recursion
$\mid\ ?\langle \mathtt{p}, \tilde{U} \rangle; T$	Input	$\mid\ T\ \mathtt{t}$	Application
$\mid\ \oplus\langle \mathtt{p}, \{l_i : T_i\}_{i \in I} \rangle$	Selection	$\mid\ \mathbf{x}$	Primitive Recursion Variable
$\mid\ \&\langle \mathtt{p}, \{l_i : T_i\}_{i \in I} \rangle$	Branching	$\mid\ \mathbf{t}$	Recursion Variable
$\mid\ \mu \mathbf{t}.T$	Recursion	$\mid\ \mathsf{end}$	End

Figure 3.12: CPS: Syntax of role and end-point types.

$\Theta; C \vdash \mathtt{p} \to \mathtt{p}' : \{l_i : G_i\}_{i \in I} \blacktriangleright \mathsf{Type}$ By assumption
$C \vdash \mathtt{p}, \mathtt{p}' \quad \forall i \in I,\ \Theta; C \vdash G_i \blacktriangleright \mathsf{Type}$ By inversion
$C \vdash \mathtt{p}, \mathtt{p}'$ is decidable By Proposition 3.4.16
$\forall i \in I,\ \Theta; C \vdash G_i \blacktriangleright \mathsf{Type}$ is decidable By i.h.
$\Theta; C \vdash \mathtt{p} \to \mathtt{p}' : \{l_i : G_i\}_{i \in I} \blacktriangleright \mathsf{Type}$ is decidable By rule

$\Theta; C \vdash \mathbf{R}\ G\ \lambda i.\lambda \mathbf{x}.G' \blacktriangleright \Pi i : \mathtt{I}.\mathsf{Type}$ By assumption
$\Theta; C \vdash G \blacktriangleright \mathsf{Type} \quad \mathbf{x} \in \mathsf{fprtv}(G') \quad \Theta, \mathbf{x} : \mathsf{Type}; C, i : \mathtt{I} \vdash G' \blacktriangleright \mathsf{Type}$ By inversion
$\Theta; C \vdash G \blacktriangleright \mathsf{Type}$ is decidable By i.h.
$\Theta, \mathbf{x} : \mathsf{Type}; C, i : \mathtt{I} \vdash G' \blacktriangleright \mathsf{Type}$ is decidable By i.h.
$\Theta; C \vdash \mathbf{R}\ G\ \lambda i.\lambda \mathbf{x}.G' \blacktriangleright \Pi i : \mathtt{I}.\mathsf{Type}$ is decidable By rule

$\Theta; C \vdash G\ \mathtt{t} \blacktriangleright \mathsf{Type}$ By assumption
$C \vdash \mathtt{t} \quad \Theta; C \vdash G \blacktriangleright \Pi i{:}\{i | i \in \mathsf{nat}, 0 \leq i \leq \mathtt{t}-1\}.\mathsf{Type}$ By inversion
$C \vdash \mathtt{t}$ is decidable By Proposition 3.4.15
$\Theta; C \vdash G \blacktriangleright \Pi i{:}\{i | i \in \mathsf{nat}, 0 \leq i \leq \mathtt{t}-1\}.\mathsf{Type}$ is decidable By i.h.
$\Theta; C \vdash G\ \mathtt{t} \blacktriangleright \mathsf{Type}$ is decidable By rule

Other cases are trivial by induction. □

3.4.4 Projection, Sorting and R-elimination

Projection A global type's projection onto the principals of roles produces types (see Figure 3.12) that capture the behaviour of roles. The constructs of role types read symmetrically to the constructs of CPS. In the case when value principals are used instead of parameterised principals in the syntax of role types, we obtain the syntax of end-point types.

The *equality between a global type principal* $\mathtt{p}$ *and role principal* $\mathtt{q}$ ensures that the set of values of $\mathtt{p}$ is a subset of the set of values of $\mathtt{q}$. The intuition underlying this design originates from the knowledge that an action performed by every process of the same role is captured by the same causality in the global type.

Definition 3.4.21 (Equality of global type and role principals)**.** A global type principal $\mathtt{p}$ is equal to a role principal $\mathtt{q}$, written $C \vdash \mathtt{p} = \mathtt{q}$ and read, "In the parameter-index typing C, global type principal $\mathtt{p}$ equals role principal $\mathtt{q}$" if

- $\mathtt{p} = \mathcal{N} \quad \mathtt{q} = \mathcal{N}$ then $C \vdash \mathcal{N} = \mathcal{N}$.
- $\mathtt{p} = \mathtt{p}'[\mathtt{i}] \quad \mathtt{q} = \mathtt{q}'[\mathtt{i}']$ and $C \vdash \mathtt{i} = \mathtt{i}' \quad C \vdash \mathtt{p}' = \mathtt{q}'$ then $C \vdash \mathtt{p}'[\mathtt{i}] = \mathtt{q}'[\mathtt{i}']$.

Figure 3.13 gives the equality relation between index expressions, transforming the equalities in the forms "parametric expression equal to index", "index equal to parametric expression" and "index equal index expression", and then checking whether, respectively, the maximum value of the (role) index is less than the (global type) parametric expression (first rule), the maximum value of the (global type) index is greater than the (role) parametric expression (forth rule), and the (global type) index expression is less than maximum value of the (role) index expression (fifth, sixth, seventh rule).

Definition 3.4.22. The minimum value of an index expression, written $C \vdash \min(\mathtt{i}) = \mathtt{t}$ and read "In the parameter-index typing C, index expression $\mathtt{i}$ has minimum value the parametric expression $\mathtt{t}$", is defined inductively on every construct of index expressions as:

- $C \vdash \min(\mathtt{t}) = \mathtt{t}$
- $C \vdash \min(i) = 0$
- $C \vdash \min(\mathrm{n} * \mathtt{i}) = \mathrm{n} * \mathcal{C} \vdash \min(\mathtt{i})$
- $C \vdash \min(\mathtt{t} \pm \mathtt{i}) = \mathtt{t} \pm C \vdash \min(\mathtt{i})$

Definition 3.4.23 (Equality of parametric expressions)**.** The equality of parametric expressions, written $C \vdash \mathtt{t} = \mathtt{t}'$ and read "In the parameter-index typing C, parametric expression $\mathtt{i}$ is equal to parametric expression $\mathtt{t}'$", is defined inductively on every construct of parametric expressions as:

- $\mathtt{t} = \mathrm{n} \quad \mathtt{t}' = \mathrm{n}$ then $C \vdash \mathrm{n} = \mathrm{n}$.

$$\frac{C \vdash i : \{i | 0 \leq i \leq \mathtt{t}'\} \quad C \vdash \mathtt{t} \leq \mathtt{t}'}{C \vdash \mathtt{t} = i} \qquad \frac{C \vdash \mathtt{t} = \mathtt{i}}{C \vdash \mathtt{t} * \mathrm{n} = \mathtt{i} * \mathrm{n}} \qquad \frac{C \vdash \mathtt{t} - \mathtt{t}' = \mathtt{i}}{C \vdash \mathtt{t} = \mathtt{t}' + \mathtt{i}} \qquad \frac{C \vdash \mathtt{t}' - \mathtt{t} = \mathtt{i}}{C \vdash \mathtt{t} = \mathtt{t}' - \mathtt{i}}$$

$$\frac{C \vdash i : \{i | 0 \leq i \leq \mathtt{t}'\} \quad C \vdash \mathtt{t} \leq \mathtt{t}'}{C \vdash i = \mathtt{t}} \qquad \frac{C \vdash i : \{i | 0 \leq i \leq \mathtt{t}\} \quad C \vdash \max(j) \leq \mathtt{t}}{C \vdash i = j}$$

$$\frac{C \vdash i : \{i | 0 \leq i \leq \mathtt{t}'\} \quad C \vdash \min(\mathtt{t} + \mathtt{j}) \geq 0 \quad C \vdash \max(\mathtt{t} + \mathtt{j}) \leq \mathtt{t}'}{C \vdash i = \mathtt{t} + \mathtt{j}}$$

$$\frac{C \vdash i : \{i | 0 \leq i \leq \mathtt{t}'\} \quad C \vdash \min(\mathtt{t} - \mathtt{j}) \geq 0 \quad C \vdash \max(\mathtt{t} - \mathtt{j}) \leq \mathtt{t}'}{C \vdash i = \mathtt{t} - j}$$

$$\frac{C \vdash \mathtt{i} = \mathtt{t}}{C \vdash n * \mathtt{i} = n * \mathtt{t}} \qquad \frac{C \vdash \mathtt{i} = \mathtt{t} + \mathtt{j}}{C \vdash n * \mathtt{i} = n * \mathtt{t} + n * \mathtt{j}} \qquad \frac{C \vdash \mathtt{i} = \mathtt{t} - \mathtt{j}}{C \vdash n * \mathtt{i} = n * \mathtt{t} - n * \mathtt{j}}$$

$$\frac{C \vdash \mathtt{i} = \mathtt{i}' - \mathtt{t}}{C \vdash \mathtt{t} + \mathtt{i} = \mathtt{i}'} \quad \frac{C \vdash \mathtt{i} = j - \mathtt{t}}{C \vdash \mathtt{t} + \mathtt{i} = j} \qquad \frac{C \vdash \mathtt{t} + \mathtt{i} = \mathtt{j}}{C \vdash n * \mathtt{t} + n * \mathtt{i} = n * \mathtt{j}}$$

$$\frac{C \vdash \mathtt{i} = \mathtt{t}' - \mathtt{t} + \mathtt{j}}{C \vdash \mathtt{t} + \mathtt{i} = \mathtt{t}' + \mathtt{j}} \quad \frac{C \vdash \mathtt{i} = \mathtt{t}' - \mathtt{t} - \mathtt{j}}{C \vdash \mathtt{t} + \mathtt{i} = \mathtt{t}' - \mathtt{j}}$$

$$\frac{C \vdash \mathtt{i} = \mathtt{t} - \mathtt{i}'}{C \vdash \mathtt{t} - \mathtt{i} = \mathtt{i}'} \quad \frac{C \vdash \mathtt{i} = \mathtt{t} - j}{C \vdash \mathtt{t} - \mathtt{i} = j} \qquad \frac{C \vdash \mathtt{t} - \mathtt{i} = \mathtt{j}}{C \vdash n * \mathtt{t} - n * \mathtt{i} = n * \mathtt{j}}$$

$$\frac{C \vdash \mathtt{i} = \mathtt{t} - \mathtt{t}' - \mathtt{j}}{C \vdash \mathtt{t} - \mathtt{i} = \mathtt{t}' + \mathtt{j}} \quad \frac{C \vdash \mathtt{i} = \mathtt{t} - \mathtt{t}' + \mathtt{j}}{C \vdash \mathtt{t} - \mathtt{i} = \mathtt{t}' - \mathtt{j}}$$

Figure 3.13: CPS: Equality between indexes.

- $\mathtt{t} = n \quad \mathtt{t}' = n$ and $C \vdash n : \mathtt{T}$ then $C \vdash n = n$.
- $\mathtt{t} = \mathtt{t}_1 \ \mathtt{op}\ \mathrm{n} \quad \mathtt{t}' = \mathtt{t}_2 \ \mathtt{op}\ \mathrm{n}$ and $C \vdash \mathtt{t}_1 = \mathtt{t}_2$ then $C \vdash \mathtt{t}_1 \ \mathtt{op}\ \mathrm{n} = \mathtt{t}_2 \ \mathtt{op}\ \mathrm{n}$.
- $\mathtt{t} = \mathrm{n}^{\mathtt{t}_1} \quad \mathtt{t}' = \mathrm{n}^{\mathtt{t}_2}$ and $C \vdash \mathtt{t}_1 = \mathtt{t}_2$ then $C \vdash \mathrm{n}^{\mathtt{t}_1} = \mathrm{n}^{\mathtt{t}_2}$.

Proposition 3.4.24 (Decidability of Parametric Equality)**.** The relation $C \vdash \mathtt{t} = \mathtt{t}'$ is decidable.

Proof. The statement that $C \vdash \mathtt{t} = \mathtt{t}'$ is decidable follows directly by induction hypothesis on every rule of the system in Definition 3.4.23. We present the most interesting cases.

Case: $C \vdash \mathrm{n} = \mathrm{n}$ is decidable since $\mathrm{n} = \mathrm{n}$ is decidable.
Case: $C \vdash \mathtt{t} \star \mathrm{n} = \mathtt{t}' \star \mathrm{n}$ is decidable since $C \vdash \mathtt{t} = \mathtt{t}'$ by induction hypothesis.

The other cases are symmetrical. □

Proposition 3.4.25 (Decidability of Indexes Equality)**.** The relation $C \vdash \mathtt{i} = \mathtt{i}'$ is decidable.

Proof. The statement that $C \vdash \mathtt{i} = \mathtt{i}'$ is decidable follows directly by induction hypothesis on every rule of the system in Figure 3.13 and Proposition 3.4.12. We present the most interesting cases.

Case: $C \vdash \mathtt{t} = \mathtt{i}$ is decidable since $C \vdash \mathtt{t} \leq \mathtt{t}'$ is decidable by Proposition 3.4.12.
Case: $C \vdash \mathtt{t} \star \mathrm{n} = \mathtt{t}' \star \mathrm{n}$ is decidable since $C \vdash \mathtt{t} = \mathtt{t}'$ by Proposition 3.4.24.
The other cases are symmetrical. □

Proposition 3.4.26 (Decidability of Principals Equality)**.** The relation $C \vdash \mathtt{p} = \mathtt{q}$ is decidable.

Proof. The statement that "$C \vdash \mathtt{p} = \mathtt{q}$ is decidable" follows directly by induction hypothesis on every rule of the system in Definition 3.4.21.

Case: $C \vdash \mathcal{N} = \mathcal{N}$ is decidable by rule.
Case: $C \vdash \mathtt{p}[\mathtt{i}] = \mathtt{q}[\mathtt{i}']$ is decidable since $C \vdash \mathtt{p} = \mathtt{q}$ is decidable by induction hypothesis and $C \vdash \mathtt{i} = \mathtt{i}'$ is decidable by Proposition 3.4.25. □

The index variables of principals in global types are different from the ones in roles, as they are bound by different binders. For this reason, we need to translate the role types being expressed from global type indexes to role ones.

Definition 3.4.27 (Substitution on principals)**.** The $\mathtt{p}'\{\mathtt{p} = \mathtt{q}\}$ operation substitutes the index variables in $\mathtt{p}'$ with expressions in terms of indexes of $\mathtt{q}$, obtained by the relation $\mathtt{p} = \mathtt{q}$ where $\mathtt{p}$ and $\mathtt{p}'$ have the same index variables, defined as:

$\mathtt{W}\{\mathtt{W}{=}\mathtt{W}\} = \mathtt{W}$
$\mathtt{p}[\mathtt{i}]\{\mathtt{p}'[\mathtt{i}']{=}\mathtt{q}[\mathtt{j}]\} = \mathtt{p}\{\mathtt{p}'{=}\mathtt{q}\}[\mathtt{i}\{\mathtt{i}'{=}\mathtt{j}\}]$

Definition 3.4.28 (Substitution on indexes)**.** The $\mathtt{i}\{\mathtt{i}' = \mathtt{j}\}$ operation substitutes the index i in $\mathtt{i}'$ and $\mathtt{i}$ with the parametric expression $\mathtt{j}'$ obtained by transforming the expression $\mathtt{i}' = \mathtt{j}$ as $i = \mathtt{j}'$:

$$\mathtt{i}\{\mathtt{i'}{=}\mathtt{j}\} = \begin{cases} \mathtt{i}\{\mathtt{j}'/i\} & \text{if } i \in \mathsf{fiv}(\mathtt{i}) \text{ and } \mathtt{i}' = \mathtt{j} \rightarrow^{\star} i = \mathtt{j}' \\ \mathtt{i} & \text{otherwise} \end{cases}$$

Definition 3.4.29. The transformation relation is defined inductively on the constructs of index expressions as:

- $i = \mathtt{j}$
- $\mathtt{n}^*\mathtt{i} = \mathtt{n}^*\mathtt{j} \rightarrow \mathtt{i} = \mathtt{j}$
- $\mathtt{t}+\ \mathtt{i} = \mathtt{j} \rightarrow \mathtt{i} = \mathtt{j}\text{-}\ \mathtt{t}$
- $\mathtt{t}\text{-}\ \mathtt{i} = \mathtt{j} \rightarrow \mathtt{i} = \mathtt{t}\text{-}\ \mathtt{j}$

Definition 3.4.30 (Projection)**.** Given global type G, principal $\mathtt{q}$, and the context C of parameter variables present in G and $\mathtt{q}$, and index variables present in $\mathtt{q}$, if $\emptyset; C \vdash G \blacktriangleright \kappa$ and $C \vdash \mathtt{q}$ then the projection of G onto $\mathtt{q}$ is defined inductively on G:

$$\mathtt{p} \to \mathtt{p}' : \langle U \rangle . G \upharpoonright \mathtt{q} = \begin{cases} !\langle \mathtt{p}'\{\mathtt{p} = \mathtt{q}\}, U\rangle(\mathtt{p}); ?\langle \mathtt{p}\{\mathtt{p}' = \mathtt{q}\}, U\rangle(\mathtt{p}'); (G \upharpoonright \mathtt{q}) & \textit{if } C \vdash \mathtt{p}{=}\mathtt{q} \textit{ and } C \vdash \mathtt{p}'{=}\mathtt{q}, \\ !\langle \mathtt{p}'\{\mathtt{p} = \mathtt{q}\}, U\rangle(\mathtt{p}); (G \upharpoonright \mathtt{q}) & \textit{if } C \vdash \mathtt{p}{=}\mathtt{q}, \\ ?\langle \mathtt{p}\{\mathtt{p}' = \mathtt{q}\}, U\rangle(\mathtt{p}'); (G \upharpoonright \mathtt{q}) & \textit{if } C \vdash \mathtt{p}'{=}\mathtt{q}, \\ G \upharpoonright \mathtt{q} & \textit{otherwise} \end{cases}$$

$$\mathtt{p} \to \mathtt{p}' : \{l_i : G_i\}_{i \in I} \upharpoonright \mathtt{q} = \begin{cases} \oplus\langle \mathtt{p}'\{\mathtt{p} = \mathtt{q}\}, \{l_i : \&\langle \mathtt{p}\{\mathtt{p}' = \mathtt{q}\}, \{l_i : G_i \upharpoonright \mathtt{q}\}_{i \in I}\rangle(\mathtt{p}')\}_{i \in I}\rangle(\mathtt{p}) & \textit{if } C \vdash \mathtt{p}{=}\mathtt{q} \textit{ and } C \vdash \mathtt{p}'{=}\mathtt{q}, \\ \oplus\langle \mathtt{p}'\{\mathtt{p} = \mathtt{q}\}, \{l_i : G_i \upharpoonright \mathtt{q}\}_{i \in I}\rangle & \textit{if } C \vdash \mathtt{p}{=}\mathtt{q}, \\ \&\langle \mathtt{p}\{\mathtt{p}' = \mathtt{q}\}, \{l_i : G_i \upharpoonright \mathtt{q}\}_{i \in I}\rangle & \textit{if } C \vdash \mathtt{p}'{=}\mathtt{q}, \\ \sqcup_{i \in I} G_i \upharpoonright \mathtt{q} & \textit{if } C \nvdash \mathtt{p}{=}\mathtt{q}, C \nvdash \mathtt{p}'{=}\mathtt{q} \\ & \forall i, j \in I . G_i \upharpoonright \mathtt{q} \bowtie G_j \upharpoonright \mathtt{q} \end{cases}$$

$$\mu \boldsymbol{t}.G \upharpoonright \mathtt{q} = \mu \boldsymbol{t}.(G \upharpoonright \mathtt{q}) \quad \boldsymbol{t} \upharpoonright \mathtt{q} = \boldsymbol{t} \quad \mathsf{end} \upharpoonright \mathtt{q} = \mathsf{end}$$

$$\mathbf{R}\ G\ \lambda i.\lambda \mathbf{x}.G' \upharpoonright \mathtt{q} = \mathbf{R}\ (G \upharpoonright \mathtt{q})\ \lambda i.\lambda \mathbf{x}.(G' \upharpoonright \mathtt{q}) \quad \mathbf{x} \upharpoonright \mathtt{q} = \mathbf{x} \quad G\ \mathtt{t} \upharpoonright \mathtt{q} = (G \upharpoonright \mathtt{q})\ \mathtt{t}$$

When a side condition does not hold, the map is undefined.

Projection is intuitive and holds some of the technical challenges of this system, which we discuss in the following paragraphs. In the role types returned, the principal in brackets attached to an action denotes the principal that performs that action, and is used to sort actions and eliminate the **R** operator from role types as we shall see later. In product global types, an *indexed principal can appear in both sides of a parameterised causality* for different values of the index variable. This occurrence is covered by the first case

of projection for message exchange and branching, which is not present in CSMS. In branching, in the case when $\mathtt{q}$ is not equal either to $\mathtt{p}$ or to $\mathtt{p}'$, all inductive projections of $\mathtt{q}$ should return an identical role type up to mergeability $\bowtie$, discussed in Section 2.6.4.

Theorem 3.4.31 (Decidability of Projection)**.** The projection of a global type onto principals is decidable.

Proof. By definition of projection on message exchange and branching constructs, the decidability of projection follows from the decidability of the relation $C \vdash \mathtt{p} = \mathtt{q}$ present in the cases of each definition. For those constructs, the proofs follows immediately from Proposition 3.4.26. For the other constructs, the statement of decidability holds by definition. □

Sorting and R-elimination Actions in the role types, returned by projection, are sorted to preserve the order of appearance in all instances of a parameterised global type. We can note from the first case of projection in the message global type, that the order of actions is not preserved; i.e., the sending action is always placed before the receiving one. However, the appearance order of actions is not only broken in the projection of a causality, but also in the sequential composition of other actions returned by projection. The reason behind this is that the order of actions depends on the order of principals performing those actions.

For example, in the Ring, the role type returned by projection of the global type, given in the introduction, onto principal $\mathtt{W}[i{+}1]$, including translation into role indexes, is:

$$\mathbf{R}\ \mathsf{end}\ \lambda j.\lambda y.!\langle \mathtt{W}[i+2], U\rangle(\mathtt{W}[n-j-1]); ?\langle \mathtt{W}[i], U\rangle(\mathtt{W}[n-j]); y\ n$$

and the action types $!\langle \mathtt{W}[i+2], U\rangle$, $?\langle \mathtt{W}[i], U\rangle$ are sorted as the Middle role firstly receives from the neighbor in its right and then sends to the one on its left. Indeed, the action $?\langle \mathtt{W}[i], U\rangle$ appears before $!\langle \mathtt{W}[i{+}2], U\rangle$, i.e., $?\langle \mathtt{W}[i], U\rangle; !\langle \mathtt{W}[i{+}2], U\rangle; \mathsf{end}$, in all instances of the global type, since a value participant will appear first in $\mathtt{W}[n{-}j]$ for bigger values of j and then in $\mathtt{W}[n{-}j{-}1]$ for smaller one; e.g. $\mathtt{W}[1]$ appears in $\mathtt{W}[n{-}j]$ for $j{=}n{-}1$ and in $\mathtt{W}[n{-}j{-}1]$ for $j{=}n{-}2$. The order of index expressions is defined on the basis that the value of i decreases in each iteration of the $\mathbf{R}$ global type.

Definition 3.4.32. The appearance order relation between two actions (`order`) is defined as the appearance order of the principals performing those actions:

$\texttt{order}(!/?^1\langle \texttt{p}_1, U\rangle(\texttt{p}'_1), !/?\langle \texttt{p}_2, U'\rangle(\texttt{p}'_2))$ iff $\texttt{order}(\texttt{p}'_1, \texttt{p}'_2)$

$\texttt{order}(\oplus/\&\langle \texttt{p}_1, \{l_i{:}T_i\}_{i\in I}\rangle(\texttt{p}'_1), \oplus/\&\langle \texttt{p}_2, \{l_i{:}T'_i\}_{i\in I'}\rangle(\texttt{p}'_2))$ iff $\texttt{order}(\texttt{p}'_1, \texttt{p}'_2)$

Definition 3.4.33. The appearance order between principals is defined as a lexicographical order over the index expressions that define them:

$\texttt{order}(\mathcal{N}[\texttt{i}_1]...[\texttt{i}_i]...[\texttt{i}_n], \mathcal{N}[\texttt{i}'_1]...[\texttt{i}'_i]...[\texttt{i}'_n])$ iff $\texttt{order}(\texttt{i}_i, \texttt{i}'_i)$ for $1 \leq i \leq n$ and

$\forall j. 1 \leq j \leq i-1.\ \mathrm{C} \vdash \texttt{i}_j = \texttt{i}'_j$ and $\mathrm{C} \nvdash \texttt{i}_i = \texttt{i}'_i$,

where the appearance order between index expressions in their canonical form is defined as:

$\texttt{order}(\texttt{t}{-}\mathrm{n}{*}i, \texttt{t}'{-}\mathrm{n}'{*}i)$ iff $\mathrm{C} \vdash \texttt{t}{-}\mathrm{n}{*}i \geq \texttt{t}'{-}\mathrm{n}'{*}i$ and

$\texttt{order}(\texttt{t}{+}\mathrm{n}{*}i, \texttt{t}'{+}\mathrm{n}'{*}i)$ iff $\mathrm{C} \vdash \texttt{t}{+}\mathrm{n}{*}i \leq \texttt{t}'{+}\mathrm{n}'{*}i$.

As mentioned above, the order of index expression is defined on the basis that the value of i decreases in each iteration of the **R** global type, resulting in the increase of values for expressions $\texttt{t}{-}\mathrm{n}{*}i$ and the decrease for $\texttt{t}{+}\mathrm{n}{*}i$. Thus, in two expressions of the form $\texttt{t}{-}\mathrm{n}{*}i$, a value will appear first in the bigger expression for bigger values of i and then in the smaller one for smaller values of i. And, in two expressions of the form $\texttt{t}{+}\mathrm{n}{*}i$, a value will appear first in the smaller expression for bigger values of i and then in the bigger one for smaller values of i. No ordering can be defined for expressions of opposite monotonicity, e.g. $\texttt{t}{-}\mathrm{n}{*}i$ and $\texttt{t}'{+}\mathrm{n}'{*}i$, as some values will appear first in the former and second in the latter, whilst some others vice versa. We can use a sorting algorithm (Merge sort, Quicksort, etc.) defined over this ordering relation to sort the actions in role types.

The **R** operator in global types iterates over parameterised causalities and defines repetitive behaviour for non index-parameterised principals. For these principals, we keep the **R** operator and the argument applied in the role types; otherwise we eliminate it by composing the two sub-types, and then later the argument. For example, the role type returned from projection of the Ring global type onto $\texttt{W}[0]$ is:

$$\mathbf{R}\ ?\langle \texttt{W}[n], U\rangle(\texttt{W}[0]); \mathsf{end}\ \lambda j.\lambda y.!\langle \texttt{W}[1], U\rangle(\texttt{W}[n-j-1]); y\ n$$

$\texttt{W}[0]$ contains one action in each sub-type and so, no sorting is performed on it. The **R** operator together with application of n carried from the global type is eliminated

[1] $!/?$ denotes either ! or ?.

by composing the two sub-types, as it does not define a repetitive behaviour for any principal; i.e. all participants in the lambda global type ($\mathtt{W}[n{-}j{-}1] \to \mathtt{W}[n-j]\colon\langle U\rangle.\mathbf{y}$) are parameterised by the index variable (j). Thus, the role type returned for type-checking of $\mathtt{W}[0]$ is $!\langle \mathtt{W}[1], U\rangle; ?\langle \mathtt{W}[n], U\rangle; \mathsf{end}$. Below, *fivr* denotes the free index variables in the bracket-principals of role types.

Definition 3.4.34. Sorting of actions, defined over `order` relation, and **R**-elimination are introduced in the function ξ, that also removes the principals in brackets, defined as:

- $\xi(!\langle \mathtt{p}, U\rangle(\mathtt{p}'); T)) = !\langle \mathtt{p}, U\rangle; \xi(T) \quad \xi(?\langle \mathtt{p}, U\rangle(\mathtt{p}'); T) = ?\langle \mathtt{p}, U\rangle; \xi(T)$
- $\xi(\oplus\langle \mathtt{p}, \{l_i : T_i\}_{i\in I}\rangle(\mathtt{p}')) = \oplus\langle \mathtt{p}, \{l_i : \xi(T_i)\}_{i\in I}\rangle$
- $\xi(\&\langle \mathtt{p}, \{l_i : T_i\}_{i\in I}\rangle(\mathtt{p}')) = \&\langle \mathtt{p}, \{l_i : \xi(T_i)\}_{i\in I}\rangle$
- $\xi(\mathbf{R}\ T\ \lambda i.\lambda \mathbf{x}.T') = \begin{cases} \mathbf{R}\ \xi(\mathtt{sort}(T))\ \lambda i.\lambda \mathbf{x}.\xi(\mathtt{sort}(T')) & i \notin \text{fivr}(T') \\ \xi(\mathtt{sort}(T'))\{\xi(\mathtt{sort}(T))/\mathbf{x}\} & \textit{otherwise} \end{cases}$
- $\xi(T\ \mathtt{t}) = \begin{cases} \xi(T)\ \mathtt{t} & \text{if } \Theta; C \vdash \xi(T) \blacktriangleright \Pi i{:}\mathtt{I}.\mathsf{Type} \\ \xi(T) & \text{otherwise} \end{cases}$
- $\xi(\mu \mathbf{t}.T) = \mu \mathbf{t}.\xi(T) \quad \xi(\mathbf{t}) = \mathbf{t} \quad \xi(\mathbf{x}) = \mathbf{x} \quad \xi(\mathsf{end}) = \mathsf{end}$

where $\Theta; C \vdash T \blacktriangleright \kappa$ ensures the well-formedness of role types in the form of a kinding judgment.

Theorem 3.4.35. Projection of G onto role principals and sorting, ordering of role types returned are decidable at the worst case in $O(p * d * n \log n)$ time complexity.

Proof. The projection algorithm has complexity $O(p * d * n)$, given: p the number of role principals, d the maximal number of index expressions in principals and n the length of the global type. The sorting algorithm (Merge sort) requires $O(p * d * n \log n)$ computational steps and **R**-elimination $O(p * n)$. Thus, the sorting algorithm dominates the cost of computation. □

3.4.5 Typing Relation

Figure 3.14 describes the typing rules for general expressions, roles and processes. The typing judgement is of the form $\Gamma; C \vdash E \rhd \tau$, read, "In the context Γ and C program E has type τ".

Definition 3.4.36 (Types and typing contexts)**.** Below, we give the formal definition of types and typing contexts:

$$\tau ::= \Delta \mid \Pi n{:}\mathtt{T}.\tau \mid \Pi i{:}\mathtt{I}.\tau \quad \Delta ::= \emptyset \mid \Delta, \mathrm{k}{:}T \quad \Gamma ::= \emptyset \mid \Gamma, u{:}\langle G\rangle \mid \Gamma, \mathbb{X}{:}S\ T \mid \Gamma, X{:}\Delta$$

The type τ represents channel and product types. The typing context Γ maps shared names, process names and type variables to types, the type Δ is a sequence of channels and their types $T@\mathtt{p}$ (see Figure 3.12). We write $dom(\Gamma)$, $dom(C)$ and $dom(\Delta)$ for the set of respectively variables, index-parameters and channels bound in Γ, C and Δ. In the case when the type has the shape $\langle G\rangle$ then we assume that G is well-formed. The rules of how a context is extended are the same as in CSMS.

The rules of appealing interest in CPS are those for program and session initiation. The other rules are identical to CSMS up to context and type constructs. Rule [TName] assigns a global type to a session identifier. Rules [TBool] and [TOr] assign the type bool to the Boolean constants **true** and **false** and to an "or" expression based on the types of its subexpressions that both must evaluate to a Boolean.

Rule $\lfloor$TFun$\rfloor$ assigns a product type to a function based on the typing of the expression: expression E must be typed under an augmented context C with mapping for parameter variable $n : \mathtt{T}$.

Rule $\lfloor$TAppF$\rfloor$ assigns a type to an application based on the type of the lambda term and the argument: the lambda term must be well-typed by a product type and the argument must fall in the set of values of the parameter $\mathtt{T}$, where min($\mathtt{T}$) represents the minimum value of n.

Definition 3.4.37. The minimum value of a parameter range is defined as:

$$\mathsf{min}(\{n \mid n \in \mathsf{nat}, n \geq \mathrm{n}\}) = \mathrm{n}$$

Rule $\lfloor$TPRec$\rfloor$ assigns a product type to a primitive recursive term based on the typing of the two sub-terms: the two sub-terms must be well-typed respectively by $\Delta\ 0$ and $\Delta\ i{+}1$ in the augmented contexts Γ and C. $\Delta\ 0$ and $\Delta\ i{+}1$ return Δ, except in the case when primitive recursion specifies a repetitive behaviour of a role; then $\Delta\ 0$ and $\Delta\ i{+}1$ return the sub-role type for type-checking of the respective sub-terms, as defined below:

Definition 3.4.38. Given $\Delta{=}\ \Delta'$, k:T. We define $\Delta\ i$ as:

$$\Delta\ i{=} \begin{cases} \Delta'\ i{,}\mathrm{k}{:}\ T\ i & \textit{if } \emptyset; C \vdash T \blacktriangleright \Pi j : \mathtt{I}.\mathsf{Type} \\ \Delta'\ i{,}\mathrm{k}{:}\ T & \textit{otherwise} \end{cases}$$

$$\Gamma, u{:}\langle G\rangle; C \vdash u{:}\ \langle G\rangle \ \lfloor \textsc{TName} \rfloor \qquad \Gamma; C \vdash \mathbf{true}, \mathbf{false}{:}\ \mathsf{bool} \ \lfloor \textsc{TBool} \rfloor$$

$$\frac{\Gamma; C \vdash e_i \triangleright \mathsf{bool}}{\Gamma; C \vdash e_1 \mathtt{or}\ e_2{:}\ \mathsf{bool}} \lfloor \textsc{TOr} \rfloor$$

$$\frac{\Gamma; C, n : \mathtt{T} \vdash E \triangleright \tau}{\Gamma; C \vdash \lambda n.E \triangleright \Pi n{:}\mathtt{T}.\tau} \lfloor \textsc{TFun} \rfloor \qquad \frac{\Gamma; C \vdash E \triangleright \Pi n{:}\mathtt{T}.\tau \quad C \vdash \mathtt{t} \geq \mathsf{min}(\mathtt{T})}{\Gamma; C \vdash E\ \mathtt{t} \triangleright \tau} \lfloor \textsc{TAppF} \rfloor$$

$$\frac{\begin{array}{c}\Gamma; C \vdash S \triangleright \Delta\ 0 \\ \Gamma, X : \Delta\ i; C, i : \mathtt{I} \vdash R \triangleright \Delta\ i+1\end{array}}{\Gamma; C \vdash \mathbf{R}\ S\ \lambda i.\lambda X.R \triangleright \Pi i{:}\mathtt{I}.\Delta} \lfloor \textsc{TPRec} \rfloor \qquad \frac{\begin{array}{c}C \vdash \mathtt{t} \quad \Gamma; C \vdash R \triangleright \\ \Pi i{:}\{i \mid i \in \mathsf{nat}, 0 \leq i \leq \mathtt{t}-1\}.\Delta\end{array}}{\Gamma; C \vdash R\ \mathtt{t} \triangleright \Delta\ \mathtt{t}} \lfloor \textsc{TAppR} \rfloor$$

$$\frac{\begin{array}{c}\Gamma; C \vdash u : \langle G\rangle \quad \emptyset; C \vdash G \blacktriangleright \mathsf{Type} \\ C \vdash \mathtt{p}_0, \mathtt{p}_1, p \quad C \vdash \mathsf{pid}(G) = \{\mathtt{p}_0, \mathtt{p}_1, p\} \\ \Gamma; C \vdash R \triangleright \Delta, y : \xi(G \upharpoonright \mathtt{p}_0)\end{array}}{\Gamma; C \vdash \bar{u}[\mathtt{p}_0, \mathtt{p}_1, p](y).R \triangleright \Delta} \lfloor \textsc{TMCast} \rfloor \qquad \frac{\begin{array}{c}\emptyset; C \vdash G \blacktriangleright \mathsf{Type} \\ \Gamma \vdash u : \langle G\rangle \quad C \vdash \mathtt{p} \\ \Gamma; C \vdash R \triangleright \Delta, y : \xi(G \upharpoonright \mathtt{p})\end{array}}{\Gamma; C \vdash u[\mathtt{p}](y).R \triangleright \Delta} \lfloor \textsc{TMAcc} \rfloor$$

$$\frac{\begin{array}{c}\Gamma; C \vdash e \triangleright S \quad C \vdash \mathtt{p} \\ \Gamma; C \vdash R \triangleright \Delta, k : T\end{array}}{\Gamma; C \vdash k!\langle \mathtt{p}, e\rangle; R \triangleright \Delta, k :!\langle \mathtt{p}, S\rangle; T} \lfloor \textsc{TSend} \rfloor \qquad \frac{C \vdash \quad \Gamma, x : S; C \vdash R \triangleright \Delta, k : T}{\Gamma; C \vdash k?\langle \mathtt{p}, x\rangle; R \triangleright \Delta, k :?\langle \mathtt{p}, S\rangle; T} \lfloor \textsc{TRcv} \rfloor$$

$$\frac{C \vdash \mathtt{p} \quad \Gamma; C \vdash R \triangleright \Delta, k : T_j \quad j \in I}{\Gamma; C \vdash k \oplus \langle \mathtt{p}, l_j\rangle; R \triangleright \Delta, k : \oplus\langle \mathtt{p}, \{l_i : T_i\}_{i \in I}\rangle} \lfloor \textsc{TSel} \rfloor$$

$$\frac{C \vdash \mathtt{p} \quad \Gamma; C \vdash R_i \triangleright \Delta, k : T_i \quad \forall i \in I}{\Gamma; C \vdash k\&\langle \mathtt{p}, \{l_i : R_i\}_{i \in I}\rangle \triangleright \Delta, k : \&\langle \mathtt{p}, \{l_i : T_k\}_{i \in I}\rangle} \lfloor \textsc{TBr} \rfloor$$

$$\frac{\Gamma, a : \langle G\rangle; C \vdash R \triangleright \Delta}{\Gamma; C \vdash (\nu a)R \triangleright \Delta} \lfloor \textsc{TNu} \rfloor \qquad \frac{\Gamma; C \vdash R \triangleright \Delta \quad \Gamma; C \vdash S \triangleright \Delta'}{\Gamma; C \vdash R \mid S \triangleright \Delta, \Delta'} \lfloor \textsc{TConc} \rfloor$$

$$\frac{\Gamma, X : \Delta; C \vdash \mathsf{Env}}{\Gamma, X : \Delta; C \vdash X \triangleright \Delta} \lfloor \textsc{TVar} \rfloor \qquad \frac{\Gamma; C \vdash \Delta \quad \Delta\ \mathsf{end\ only}}{\Gamma; C \vdash \mathbf{0} \triangleright \Delta} \lfloor \textsc{TInact} \rfloor$$

$$\frac{\Gamma; C \vdash e \triangleright \mathsf{bool} \qquad \Gamma; C \vdash R \triangleright \Delta \qquad \Gamma; C \vdash S \triangleright \Delta}{\Gamma; C \vdash \mathsf{if}\ e\ \mathsf{then}\ R\ \mathsf{else}\ S \triangleright \Delta} \lfloor \textsc{TIf} \rfloor \qquad \frac{\Gamma; C \vdash R \triangleright \Delta \qquad \Delta <: \Delta'}{\Gamma; C \vdash R \triangleright \Delta'} \lfloor <: \rfloor$$

$$\frac{\Gamma; C \vdash R \triangleright \Delta, s[\hat{\mathtt{p}}_1] : T_1, ..., s[\hat{\mathtt{p}}_n] : T_n}{\Gamma; C \vdash (\nu s)R \triangleright \Delta} \lfloor \textsc{TCRes} \rfloor$$

$$\frac{\Gamma; C \vdash \tilde{e}{:}\ \tilde{S} \qquad \Delta\ \mathsf{end\ only}}{\Gamma, X{:}\tilde{S}\tilde{T}; C \vdash X\langle \tilde{e}, \tilde{k}\rangle \triangleright \Delta, \tilde{k} : \tilde{T}} \lfloor \textsc{TVar} \rfloor \qquad \frac{\begin{array}{c}\Gamma, X{:}\tilde{S}\tilde{T}, \tilde{y} : \tilde{S}; C \vdash R \triangleright \tilde{y'} : \tilde{T} \\ \Gamma, X{:}\tilde{S}\tilde{T}; C \vdash R' \triangleright \Delta\end{array}}{\Gamma; C \vdash \mathsf{def}\ X(\tilde{y}, \tilde{y'}) = R\ \mathsf{in}\ R' \triangleright \Delta} \lfloor \textsc{TDef} \rfloor$$

Figure 3.14: CPS: Typing system for general expressions, roles and processes.

The following definition defines how the variable X is typed by $\Delta\ i$, where the latter is denoted by $\mathbf{R}\ T\ \lambda j.\lambda x.T'\ i$ in the $\lfloor$TPRec$\rfloor$ rule.

Definition 3.4.39. An application of an index variable to a primitive recursive end-point type is equal by definition to:

$$\mathbf{R}\ T\ \lambda j.\lambda x.T'\ i{+}1 \triangleq T'\{i/j\}\{\mathbf{R}\ T\ \lambda j.\lambda x.T'\ i/x\}$$

Rule $\lfloor$TAppR$\rfloor$ assigns a type to an application of a parametric expression to primitive recursion based on the typing of the primitive recursive term and argument: the primitive recursive term must be well-typed by a product type where the type of the index ($\mathtt{I}$) has for maximum value a predecessor of the argument, and the argument must evaluate to a natural number.

Rules $\lfloor$TMCast$\rfloor$ and $\lfloor$TMAcc$\rfloor$ assign a type to a role prefixed by a session initiation construct based on the typing of the role: the session identifier must be typed by a global type G and the process R must be typed by (1) the projection of the global type onto the participant of the prefix ($\mathtt{1}$ in case of [MCast]) and (2) sorting and **R**-elimination of role types, where the sets of participants of R and G are the same ensured by the relation $\vdash \mathsf{pid}(G) = \{\mathtt{p}_0, \mathtt{p}_1, p\}$. The equality relation between the two sets of principals is defined over the mathematical definition of parameterised list of principals discussed in Section 3.4.1. The well-formedness of the list of principals in R is defined as:

Definition 3.4.40. A list of principals is well-formed if every component of the list is well-formed:

$C \vdash \mathtt{p}_1, ..., \mathtt{p}_n$ if $C \vdash \mathtt{p}_1 .... C \vdash \mathtt{p}_n$ and
$C \vdash \mathbf{R}\ p\ \lambda i.\lambda X.p'\ \mathtt{t}$ if $C \vdash p \quad C \vdash \mathtt{t} \quad C, i : \{i | 0 \leq i \leq \mathtt{t} - 1\} \vdash p'$.

All the conditions to invoke projection are ensured by the rules (well-formedness of global types and role principal). The type assigned to the prefixed role, Δ without the typing of the process, means that the role of session u does not evaluate to any interactions at this point. The metavariable Δ in the typing of R and the prefixed role denotes the typing of other possible sessions running in them. Below, we give the definition of the principal set equality relation.

Definition 3.4.41. The equality set relation between principals of global types and roles is defined as the inclusion relation of one set to the other and vice versa:

$$C \vdash \mathsf{pid}(G) = \{\mathtt{p}_0, \mathtt{p}_1, p\}\ \textit{if}\ C \vdash \mathsf{pid}(G) \subseteq \{\mathtt{p}_0, \mathtt{p}_1, p\}\ \textit{and}\ C \vdash \{\mathtt{p}_0, \mathtt{p}_1, p\} \subseteq \mathsf{pid}(G).$$

Definition 3.4.42. The inclusion relation of global type principals to role principals is defined as the membership relation of each participant of the former set to the latter, given the range of values of the global type participant:

$$C \vdash \{\mathtt{p}'_0, \mathtt{p}'_1, ..., \mathtt{p}'_n\} \subseteq \{\mathtt{p}_0, \mathtt{p}_1, p\} \textit{ if } \forall \mathtt{p} \in \{\mathtt{p}'_0, \mathtt{p}'_1, ..., \mathtt{p}'_n\}. C; \mathtt{range}(\mathtt{p}) \vdash \mathtt{p} \in \{\mathtt{p}_0, \mathtt{p}_1, p\}.$$

Definition 3.4.43. The inclusion relation of role principals to global type principals is defined as the membership relation of each participant of the former set, including the parameterised list of principals, to the latter:

$$C \vdash \{\mathtt{p}_0, \mathtt{p}_1, p\} \subseteq \{\mathtt{p}'_0, \mathtt{p}'_1, ..., \mathtt{p}'_n\} \textit{ if } C \vdash \mathtt{p}_0, \mathtt{p}_1, p \in \{\mathtt{p}'_0, \mathtt{p}'_1, ..., \mathtt{p}'_n\}.$$

Definition 3.4.44. The membership relation of a principal to a set of principals is defined as the inclusion of participant range to the participants set range:

(1) $C; [a..b] \vdash \mathtt{p} \in \{\mathtt{p}_1, ..., \mathtt{p}_n\}$ *if* $\mathtt{intersection}(C, [a..b], \mathtt{p}, \mathtt{p}_1) = [a..b]$ or

(2) $C; [a_1..b_1]..[a_j..b_j] \vdash \mathtt{p} \in \{\mathtt{p}_1, ..., \mathtt{p}_n\}$ *if* $\mathtt{intersection}(C, [a_1..b_1]..[a_j..b_j], \mathtt{p}, \mathtt{p}_1) = [a'_1..b'_1]..[a'_i..b'_i]$ *and* $C; [a'_1..b'_1]..[a'_i..b'_i] \vdash \mathtt{p} \in \{\mathtt{p}_2, ..., \mathtt{p}_n\}$

Definition 3.4.45. The membership relation of a principal to a parameterised list of principals is defined as the inclusion of participant range to the participants of the list, given the range of the parameterised list index:

$C; [a_1..b_1]..[a_j..b_j] \vdash \mathtt{p} \in \{\mathbf{R}\ p\ \lambda i.\lambda x.p'\ \mathtt{t}\}$ *if* $C, i : \{i | 0 \leq i \leq \mathtt{t} - 1\}; [a_1..b_1]..[a_j..b_j] \vdash \mathtt{p} \in \{p, p'\}$

Definition 3.4.46. The membership relation of the paramaterised list of principals to the global types principals is defined as the inclusion relation of the principals in the parameterised list into the principals of the global type, given the range of the parameterised list index:

$C \vdash \mathbf{R}\ p\ \lambda i.\lambda x.p'\ \mathtt{t} \in \{\mathtt{p}'_0, \mathtt{p}'_1, ..., \mathtt{p}'_n\}$ *if* $C, i : \{i | 0 \leq i \leq \mathtt{t} - 1\} \vdash \{p, p'\} \subseteq \{\mathtt{p}'_0, \mathtt{p}'_1, ..., \mathtt{p}'_n\}$.

Definition 3.4.47. The `intersection` operation finds the set of values of $\mathtt{p}$ that overlap with that of $\mathtt{p}'$:

$\mathtt{intersection}(C, [a_1..b_1]..[a_j..b_j], \mathtt{p}, \mathtt{p}') = [a_1..b_1]..[a_j..b_j] \backslash [a'_1..b'_1]...[a'_k..b'_k]$ *if* $C \vdash \mathtt{p} = \mathtt{p}'$ *where* $\mathtt{range}(\mathtt{p}') = [a''..b'']$ *and* $[a'_1..b'_1]...[a'_k..b'_k] = [a_1..b_1]..[a_j..b_j] \cap [a''..b'']$.

Proposition 3.4.48 (Decidability Principal Set Equality)**.** The relation $C \vdash \mathsf{pid}(G) = \{\mathtt{p}_0, \mathtt{p}_1, p\}$ is decidable.

Proof. Follows immediately by Definitions 3.4.41-3.4.47. □

Rules ⌊TSend⌋ and ⌊TRcv⌋ assign a type to a role respectively prefixed by a sending and receiving action based on the typing of the subrole: for the sending prefix, the expressions sent $\tilde{e}$ must evaluate to values of type $\tilde{U}$, the receiving participant $\mathtt{p}$ in the prefix must match the one in the role type and R must be well-typed by an end-point type T, while for the receiving prefixes R must be well-typed by a type T over an extended context with the place-holders of values.

Rules ⌊TSel⌋ and ⌊TBr⌋ assign a type to respectively a role prefixed by selection and branching process based on the typing of the subroles: for selection, R must be well-typed by a role type T_j such that T_j is prefixed by the label selected, while for branching every role labeled must be well-typed by a role type T_i. The resulting type is the set of label and role type prefixed by the channels performing the action and symbols $\oplus, \&$ denoting respectively the "selection" and "branching" operation.

Rule ⌊TNu⌋ is standard. Rule ⌊TConc⌋ assigns a type to two roles composed in parallel based on their respective typing: each subrole must be well-typed. Rules ⌊TIf⌋, ⌊<:⌋, ⌊TVar⌋, ⌊TInact⌋ are standard. Subtyping over context typing is defined over role types as:

Definition 3.4.49. $\Delta <: \Delta'$ if for $k \in dom(\Delta)$ and $k \in dom(\Delta')$ then $\Delta(k) <: \Delta'(k)$ or $\Delta(k) = \Delta'(k)$.

Subtyping over role types is defined as in standard session subtyping by Gay and Hole (2005); Honda et al. (2008b).

Definition 3.4.50. The subtyping over role types, denoted <:, is given as the maximal fixed point of the function $\mathbf{S}$ that maps each binary relation R on role types as regular trees to $\mathbf{S}(R)$ given as:

- If TRT' then $!\langle \mathtt{p}, \tilde{U}\rangle; T\ \mathbf{S}(R)\ !\langle \mathtt{p}, \tilde{U}\rangle; T'$ and $?\langle \mathtt{p}, \tilde{U}\rangle; T\ \mathbf{S}(R)\ ?\langle \mathtt{p}, \tilde{U}\rangle; T'$.
- If $T_i R T_i'$ for each $i \in I \subset J$ then both $\oplus\langle \mathtt{p}, \{l_i : T_i\}_{i\in I}\rangle \mathbf{S}(R) \oplus \langle \mathtt{p}, \{l_i : T_i'\}_{i\in J}\rangle$ and $\&\langle \mathtt{p}, \{l_i : T_i\}_{i\in J}\rangle \mathbf{S}(R) \& \langle \mathtt{p}, \{l_i : T_i'\}_{i\in I}\rangle$.

Rule ⌊TCRes⌋ assigns a type to process prefixed by a channel restriction based on the typing of the subprocess: subprocess R must be well-typed by the union of all end-points $T_1, ..., T_n$. Rules ⌊TVar⌋ and ⌊TDef⌋ are standard.

Conjecture 3.4.51 (Decidability of Typing)**.** The typing relation $\Gamma; C \vdash E \triangleright \tau$ is decidable.

Although, we do not have a formal proof, we believe the type system is decidable. Indeed, the type system benefits from the decidability for (1) principals set equality (Proposition 3.4.48), (2) projection (Theorem 3.4.35), (3) kinding of global types (Theorem 3.4.20) and (4) principals well-formedness (Proposition 3.4.16). We leave the proof of decidability of the type system for future work.

Typing queue Below we give the message types of queues

$$\begin{array}{rrcll} \text{Message} & T & ::= & !\langle \hat{\mathtt{p}}, U\rangle & \textit{message send} \\ & & \mid & \oplus\langle \hat{\mathtt{p}}, l\rangle & \textit{message selection} \\ & & \mid & T; T' & \textit{message sequence} \end{array}$$

where $!\langle \hat{\mathtt{p}}, U\rangle$ expresses the communication to $\hat{\mathtt{p}}$ of a value of type U, $\oplus\langle \hat{\mathtt{p}}, l\rangle$ expresses the communication to $\hat{\mathtt{p}}$ of a label l and $T; T'$ represents sequencing of message types (we assume associativity for ;). The typing of queue is similar as in Bettini et al. (2008) with the difference that context typing contains also the index-parameter typing. The empty queue has an empty mapping from channels to message types. A message adds an output type to the type of the channel in the mapping; we present the rule of a adding a value message since the rule for adding a label is similar:

$$\frac{\Gamma; C \vdash h \triangleright \Delta \quad \Gamma; C \vdash \triangleright S}{\Gamma; C \vdash s : h \cdot (\hat{\mathtt{p}}, \hat{\mathtt{q}}, v) \triangleright \Delta; \{s[\hat{\mathtt{p}}] : !\langle \hat{\mathtt{q}}, S\rangle\}} \lfloor \text{TRSend} \rfloor$$

where ; is defined as:

$$\Delta; \{s[\hat{\mathtt{p}}] : T\} = \begin{cases} \Delta', s[\hat{\mathtt{p}}] : T'; T & \text{if } \Delta = \Delta', s[\hat{\mathtt{p}}] : T' \\ \Delta, s[\hat{\mathtt{p}}] : T & \text{otherwise} \end{cases}$$

The typing of processes in parallel with queues is defined as in Bettini *et al.*:

$$\frac{\Gamma; C \vdash P \rhd \Delta \quad \Gamma; C \vdash Q \rhd \Delta'}{\Gamma; C \vdash P \mid Q \rhd \Delta \star \Delta'} \lfloor \text{TRConc} \rfloor$$

where $\star$ is defined as:

$$T \star T' = \begin{cases} T; T' & \text{if T is a message type,} \\ T'; T & \text{if T' is a message type,} \\ \text{undefined} & \text{otherwise} \end{cases}$$

and in the session environment is defined as $\Delta \star \Delta' = \Delta \setminus dom(\Delta') \cup \Delta' \setminus dom(\Delta) \cup \{c : T \star T' | c : T \in \Delta \text{ and } c : T' \in \Delta'\}$.

3.4.6 Typing Communication Patterns

We type the programs of the Ring, Tree and Star, given in sections 3.1 and 3.2, with focus on sorting and **R**-elimination in the Ring, projection and principals set equality in the Tree, and rule $\lfloor$TPRec$\rfloor$ in the Star.

Ring The main program is typed by $\Pi n{:}\{n \mid n{\in}\mathsf{nat}, n{\geq}2\}.\Delta$, restricting only with a lower bound the set of natural numbers that can be applied. The **R** term is typed by $\Pi i{:}\{i \mid i{\in}\mathsf{nat}, 0{\leq}i{\leq}n{-}2\}.\Delta$ as each role is typed by Δ and the set $\{i \mid i{\in}\mathsf{nat}, 0{\leq}i{\leq}n{-}2\}$ is well-defined on the assumption that $n \geq 2$. Projection of the global type onto the principals $\mathtt{W}[0], \mathtt{W}[i{+}1]$ and $\mathtt{W}[n]$ returns the following role-types:

$$\begin{aligned} &\mathbf{R}\ ?\langle \mathtt{W}[n], U\rangle(\mathtt{W}[0]); \mathsf{end}\ \lambda j.\lambda y.!\langle \mathtt{W}[1], U\rangle(\mathtt{W}[n-j-1]); y\ n, \\ &\mathbf{R}\ \mathsf{end}\ \lambda j.\lambda y.!\langle \mathtt{W}[i+2], U\rangle(\mathtt{W}[n-j-1]); ?\langle \mathtt{W}[i], U\rangle(\mathtt{W}[n-j]); y\ n \text{ and} \\ &\mathbf{R}\ !\langle \mathtt{W}[0], U\rangle(\mathtt{W}[n]); \mathsf{end}\ \lambda j.\lambda y.?\langle \mathtt{W}[n-1], U\rangle(\mathtt{W}[n-j-1]); y\ n. \end{aligned}$$

where for each action type, the principal in brackets denotes the principal in the global type that performs that action. The role types are translated in terms of roles index (i); e.g. the action $!\langle \mathtt{W}[n{-}j]\{n{-}j{-}1 = i{+}1\}, U\rangle(\mathtt{W}[n-j-1])$ in the role type of $\mathtt{W}[i{+}1]$ is translated to $!\langle \mathtt{W}[i+2], U\rangle(\mathtt{W}[n{-}j{-}1])$ by substituting in $n{-}j$, j with $n{-}2{-}i$. $\mathtt{W}[n]$ contain one action in each sub-type and so, no sorting is performed on it. The **R** operator together with application of n carried from the global type is eliminated by composing the two sub-types, as it does not define a repetitive behaviour for any principal; i.e. all participant in the

lambda global type ($\mathtt{W}[n{-}j{-}1] \to \mathtt{W}[n-j] : \langle U \rangle.\mathbf{y}$) are parameterised by the index variable (j). Thus, the role type returned for type-checking of $\mathtt{W}[n]$ is $?\langle \mathtt{W}[n{-}1], U\rangle; !\langle \mathtt{W}[0], U\rangle; \mathsf{end}$. The role type of $\mathtt{W}[0]$ is discussed in Section 3.4.4.

The role type of $\mathtt{W}[i{+}1]$ has a more sophisticated structure than the other principals. It has two actions in the lambda role type that are sorted to preserve the order of appearance in all the instances of the global type $?\langle \mathtt{W}[i], U\rangle; !\langle \mathtt{W}[i{+}2], U\rangle; \mathsf{end}$. The ordering of actions was explained in Section 3.4.4. This is the role type returned by the ξ function (sorting, and **R**-elimination), given the role type returned by projection of the global type:

$$\mathbf{R}\ \mathtt{W}[n] \to \mathtt{W}[0] : \langle U \rangle.\mathsf{end}\ \lambda j.\lambda \mathbf{y}.\mathtt{W}[n{-}j{-}1] \to \mathtt{W}[n-j] : \langle U \rangle.\mathsf{end}\ \mathrm{n}$$

and so, well-typing a session of n participants of a global type that specifies the Ring of length 1 as discussed in Section 3.4.3. Thus, the well-formedness rules of global type must ensure that a product global type defines a parameterised communication pattern. Type-checking of the roles with the role types, obtained by projection is straightforward.

Tree Typing manually all the five roles is tedious; thus we limit the description of typing for roles of $\mathtt{W}[0]$ and $\mathtt{W}[2{*}i{+}1]$. Type-checking the role of $\mathtt{W}[0]$ by the type $!\langle \mathtt{W}[1], U\rangle; !\langle \mathtt{W}[2], U\rangle; \mathsf{end}$ is straightforward. The interesting part of typing is checking whether the relation $C \vdash \{\mathtt{W}[j], \mathtt{W}[2{*}j{+}1], \mathtt{W}[2{*}j{+}2]\} = \{\mathtt{W}[0], \mathtt{W}[1], \mathtt{W}[2^{n+1}{-}2], \mathtt{W}[i{+}2]\}$ holds in the rule $\lfloor$TMCast$\rfloor$. Two principal sets are equal only if each set is a subset of the other. The subset relation $C \vdash A{\subseteq}B$ holds only if each element of A is an element of B. The membership relation of a principal $\mathtt{p}$ in a set A holds only if the set of values of the principal is contained in the sets of values of all the members of A. One can easily verify that the set of values of $\mathtt{W}[\mathrm{j}]$ ($0{\leq}j{\leq}2^n{-}1$) is contained in the set of values of $\{\mathtt{W}[0], \mathtt{W}[1], \mathtt{W}[i{+}2]\}$ ($0{\leq}i{\leq}\ 2^{n+1}{-}5$), $\mathtt{W}[2{*}\mathrm{j}{+}1]$ is contained in $\{\mathtt{W}[1],\ \mathtt{W}[\mathrm{i}{+}2]\}$ and $\mathtt{W}[2{*}\mathrm{j}{+}2]$ in $\{\mathtt{W}[\mathrm{i}{+}2],\ \mathtt{W}[2^{n+1}{-}2]\}$. The check of $C{\vdash}\{\mathtt{W}[0], \mathtt{W}[1], \mathtt{W}[i{+}2], \mathtt{W}[2^{n+1}{-}2]\}{\subseteq}\{\mathtt{W}[j], \mathtt{W}[2{*}j{+}1],\ \mathtt{W}[2{*}j{+}2]\}$ is similar but tedious so we leave it to the curious reader.

Projection of the global type onto $\mathtt{W}[2{*}i{+}1]$ recursively checks on each causality for principals equal to $\mathtt{W}[2{*}i{+}1]$. For example, $C{\vdash}\mathtt{W}[j]{=}\mathtt{W}[2{*}i{+}1]$ holds because j can be represented as an index expression in terms of i and $C{\vdash}\max(2{*}i{+}1){\leq}\max(j)$ holds (affirming the knowledge that an action performed by every process of the same role is captured by the same causality in the global type), since $C{\vdash}2{*}(2^{n-1}{-}2){+}1{\leq}2^n{-}2$ for $\max(i){=}2^{n-1}{-}2$ and $\max(j){=}2^n{-}\ 2$. Thus, the action obtained is $!\langle \mathtt{W}[2{*}j{+}1]\{\mathtt{W}[j]{=}\mathtt{W}[2{*}i{+}1]\}, U\rangle$ which is translated in terms of i by substituting j in $2{*}j{+}1$ with $2{*}i{+}1$, returning $!\langle \mathtt{W}[4{*}i{+}3], U\rangle$.

Principal $\mathtt{W}[2{*}j{+}2]$ is not equal to $\mathtt{W}[2{*}i{+}1]$ as j cannot be represented as an index expressions in terms of i ($j{=}i{-}1/2$ is not an index expression $\mathtt{i}$). The role type returned following projection, sorting and $\mathbf{R}$-elimination is $!\langle \mathtt{W}[4{*}i{+}3], U\rangle; !\langle \mathtt{W}[4{*}i{+}4], U\rangle; ?\langle \mathtt{W}[i], U\rangle; \mathsf{end}$.

Star Projection of the Star global type, given in Section 3.2, onto principal $\mathtt{W}[0]$ returns the following role type:

$$\mathbf{R}\ \mathsf{end}\ \lambda j.\lambda y.!\langle \mathtt{W}[j+1], U\rangle(\mathtt{W}[0]).y\ n$$

The $\mathbf{R}$ operator is not eliminated by the ξ function as the index $j \notin \mathrm{fivr}(\mathtt{W}[0])$, returning the same role type from projection. From rule [TPRec], the sub-term end is typed by $\Delta\ 0$ where $\Delta\ 0 = \mathbf{R}\ \mathsf{end}\ \lambda j.\lambda y.!\langle \mathtt{W}[j+1], U\rangle(\mathtt{W}[0]).y\ 0 = \mathsf{end}$ and the recursive term $y!\langle \mathtt{W}[j+1], f(j+1)\rangle; Y$ is typed by $\Delta\ j+1$ where $\Delta\ j+1 = \mathbf{R}\ \mathsf{end}\ \lambda j.\lambda y.!\langle \mathtt{W}[j+1], U\rangle(\mathtt{W}[0]).y\ j+1 = \langle \mathtt{W}[j+1], U\rangle(\mathtt{W}[0]).\Delta j$ by Definition 3.4.39.

3.4.7 Properties of the Typing Relation

We prove type preservation and communication-safety of this system, the statements of which were presented in Section 2.6.5. Reduction over session typings are introduced below, representing interactions in processes. We assume well-formedness of types.

$$\{s[\hat{\mathtt{p}}] :!\langle \hat{\mathtt{q}}, U\rangle; T, s[\hat{\mathtt{q}}] :?\langle \hat{\mathtt{p}}, U\rangle; T'\} \Rightarrow \{s[\hat{\mathtt{p}}] : T, s[\hat{\mathtt{q}}] : T'\} \quad \text{[TR-Ex]}$$

$$\{s[\hat{\mathtt{p}}] : \oplus\langle \hat{\mathtt{q}}, \{l_j : T_j\}_{j\in I}\rangle, s[\hat{\mathtt{q}}] : \&\langle \hat{\mathtt{p}}, \{l_k : T'_k\}_{k\in J}\rangle\} \Rightarrow \{s[\hat{\mathtt{p}}] : T_i, s[\hat{\mathtt{q}}] : T'_i\} \quad \text{[TR-MultL]}$$

$$\{s[\hat{\mathtt{p}}] : T, ...\} \Rightarrow \{s[\hat{\mathtt{p}}] : T\{\mathrm{n}/n\}, ..., \} \quad \text{[TR-Apply]}$$

$$\Delta \Rightarrow \Delta' \text{ if } \{s[\hat{\mathtt{p}}] : T_1, s[\hat{\mathtt{q}}] : T_2\} \Rightarrow \{s[\hat{\mathtt{p}}] : T'_1, s[\hat{\mathtt{q}}] : T'_2\} \text{ where } \{s[\hat{\mathtt{p}}] : T_1, s[\hat{\mathtt{q}}] : T_2\} \subseteq \Delta \quad \text{[TR-Context]}$$

Rules [TR-Ex], [TR-MultL] and [TR-Context] read symmetrically to the rules in CSMS, respectively, [TR-Mult], [TR-MultL] and [TR-Context]. Rule [TR-Apply] represents the action of applying a natural number to a parameter bound in the context of typing.

Lemma 3.4.52 (Weakening Lemma)**.** 1. Let $\Gamma; C \vdash E \triangleright \tau$. If $a \notin dom(\Gamma)$, then $\Gamma, a : \langle G\rangle; C \vdash E \triangleright \tau$.

2. Whenever $\Gamma; C \vdash R \triangleright \Delta$ is derivable then its weakening, $\Gamma; C \vdash R \triangleright \Delta, \Delta'$ for disjoint Δ' where Δ' contains only empty type contexts and for types end, is also derivable.
3. Let $\Gamma; C \vdash E \triangleright \tau$. If $X \notin dom(\Gamma)$, then $\Gamma, X : \tilde{S}\tilde{T}; C \vdash E \triangleright \tau$.

Proof. (1) is trivial by induction on the derivation of $\Gamma; C \vdash E \triangleright \tau$. We present the most appealing cases, including the most difficult.

$\Gamma, u : \langle G' \rangle; C \vdash u : \langle G' \rangle$ and $a \notin dom(\Gamma, u : \langle G' \rangle)$ By assumption
$\Gamma, u : \langle G' \rangle, a : \langle G \rangle; C \vdash u : \langle G' \rangle$ By rule [TName]

$\Gamma; C \vdash \lambda n.E \triangleright \Pi n : \mathtt{T}.\tau$ and $a \notin dom(\Gamma)$ By assumption
$\Gamma; C, n : \mathtt{T} \vdash E \triangleright \tau$ and $a \notin dom(\Gamma)$ By inversion
$\Gamma, a : \langle G \rangle; C, n : \mathtt{T} \vdash E \triangleright \tau$ By i.h.
$\Gamma, a : \langle G \rangle; C \vdash \lambda n.E \triangleright \Pi n : \mathtt{T}.\tau$ By rule [TFun]

$\Gamma; C \vdash E\ \mathtt{t} \triangleright \tau$ and $a \notin dom(\Gamma)$ By assumption
$\Gamma; C \vdash E \triangleright \Pi n : \mathtt{T}.\tau \quad C \vdash \mathtt{t} \geq \mathsf{min}(\mathtt{T})$ and $a \notin dom(\Gamma)$ By inversion
$\Gamma, a : \langle G \rangle; C \vdash E \triangleright \Pi n : \mathtt{T}.\tau \quad C \vdash \mathtt{t} \geq \mathsf{min}(\mathtt{T})$ By i.h.
$\Gamma, a : \langle G \rangle; C \vdash E\ \mathtt{t} \triangleright \tau$ By rule [TAppF]

$\Gamma; C \vdash \mathbf{R}\ S\ \lambda i.\lambda X.R \triangleright \Pi i{:}\mathtt{I}.\Delta$ and $a \notin dom(\Gamma)$ By assumption
$\Gamma; C \vdash S \triangleright \Delta\ 0 \quad \Gamma, X : \Delta\ i; C, i : \mathtt{I} \vdash R \triangleright \Delta\ i+1$ and $a \notin dom(\Gamma)$ By inversion
$\Gamma, a : \langle G \rangle; C \vdash S \triangleright \Delta\ 0 \quad \Gamma, X : \Delta\ i, a : \langle G \rangle; C, i : \mathtt{I} \vdash R \triangleright \Delta\ i+1$ By i.h.
$\Gamma, a : \langle G \rangle; C \vdash \mathbf{R}\ S\ \lambda i.\lambda X.R \triangleright \Pi i{:}\mathtt{I}.\Delta$ By rule [TPRec]

$\Gamma; C \vdash R\ \mathtt{t} \triangleright \Delta\ \mathtt{t}$ and $a \notin dom(\Gamma)$ By assumption
$C \vdash \mathtt{t} \quad \Gamma; C \vdash R \triangleright \Pi i{:}\{i \mid i{\in}\mathsf{nat}, 0{\leq}i{\leq}\mathtt{t}{-}1\}.\Delta$ and $a \notin dom(\Gamma)$ By inversion
$C \vdash \mathtt{t} \quad \Gamma, a : \langle G \rangle; C \vdash R \triangleright \Pi i{:}\{i \mid i{\in}\mathsf{nat}, 0{\leq}i{\leq}\mathtt{t}{-}1\}.\Delta$ By i.h.
$\Gamma, a : \langle G \rangle; C \vdash R\ \mathtt{t} \triangleright \Delta\ \mathtt{t}$ By rule [TAppR]

$\Gamma; C \vdash \bar{u}[\mathtt{p}_0, \mathtt{p}_1, p](y).R \triangleright \Delta$ and $a \notin dom(\Gamma)$ By assumption
$\Gamma; C \vdash u : \langle G' \rangle \quad \emptyset; C \vdash G \blacktriangleright \mathsf{Type} \quad C \vdash \mathtt{p}_0, \mathtt{p}_1, p \quad C \vdash \mathsf{pid}(G') = \{\mathtt{p}_0, \mathtt{p}_1, p\}$
$\Gamma; C \vdash R \triangleright \Delta, y : \xi(G' \upharpoonright \mathtt{p}_0)$ and $a \notin dom(\Gamma)$ By inversion

$\Gamma, a : \langle G \rangle; C \vdash u : \langle G' \rangle \quad \emptyset; C \vdash G \blacktriangleright \mathsf{Type} \quad C \vdash \mathtt{p}_0, \mathtt{p}_1, p \quad C \vdash \mathsf{pid}(G') = \{\mathtt{p}_0, \mathtt{p}_1, p\}$
$\Gamma, a : \langle G \rangle; C \vdash R \rhd \Delta, y : \xi(G' \upharpoonright \mathtt{p}_0)$ By i.h.
$\Gamma, a : \langle G \rangle; C \vdash \bar{u}[\mathtt{p}_0, \mathtt{p}_1, p](y).R \rhd \Delta$ By rule [TMCast]

Case [TMAcc] is symmetric.

$\Gamma; C \vdash k!\langle \mathtt{p}, e \rangle; R \rhd \Delta, k :!\langle \mathtt{p}, S \rangle; T$ and $a \notin dom(\Gamma)$ By assumption
$\Gamma; C \vdash e \rhd S \quad C \vdash \mathtt{p} \quad \Gamma; C \vdash R \rhd \Delta, k : T$ and $a \notin dom(\Gamma)$ By inversion
$\Gamma, a : \langle G \rangle; C \vdash e \rhd S \quad C \vdash \mathtt{p} \quad \Gamma, a : \langle G \rangle; C \vdash R \rhd \Delta, k : T$ By i.h.
$\Gamma, a : \langle G \rangle; C \vdash k!\langle \mathtt{p}, e \rangle; R \rhd \Delta, k :!\langle \mathtt{p}, S \rangle; T$ By rule [TSend]

Cases [TRcv], [TSel], [TBr] are symmetric.

$\Gamma; C \vdash (\nu b)R \rhd \Delta$ and $a \notin dom(\Gamma)$ By assumption
$\Gamma, b : \langle G' \rangle; C \vdash R \rhd \Delta$ and $a \notin dom(\Gamma)$ By inversion
$\Gamma, b : \langle G' \rangle, a : \langle G \rangle; C \vdash R \rhd \Delta$ By i.h.
$\Gamma, a : \langle G \rangle; C \vdash (\nu b)R \rhd \Delta$ By rule [TNu]

$\Gamma; C \vdash R \mid S \rhd \Delta, \Delta'$ and $a \notin dom(\Gamma)$ By assumption
$\Gamma; C \vdash R \rhd \Delta \quad \Gamma; C \vdash S \rhd \Delta'$ and $a \notin dom(\Gamma)$ By inversion
$\Gamma, a : \langle G \rangle; C \vdash R \rhd \Delta \quad \Gamma, a : \langle G \rangle; C \vdash S \rhd \Delta'$ By i.h.
$\Gamma, a : \langle G \rangle; C \vdash R \mid S \rhd \Delta, \Delta'$ By rule [TConc]

Cases [TVar], [TInact], [TIf], [<:] are trivial.

$\Gamma; C \vdash (\nu s)R \rhd \Delta$ and $a \notin dom(\Gamma)$ By assumption
$\Gamma; C \vdash R \rhd \Delta, s[\hat{\mathtt{p}}_1] : T_1, ..., s[\hat{\mathtt{p}}_n] : T_\mathrm{n}$ and $a \notin dom(\Gamma)$ By inversion
$\Gamma, a : \langle G \rangle; C \vdash R \rhd \Delta, s[\hat{\mathtt{p}}_1] : T_1, ..., s[\hat{\mathtt{p}}_n] : T_\mathrm{n}$ By i.h.
$\Gamma, a : \langle G \rangle; C \vdash (\nu s)R \rhd \Delta$ By rule [TCres]

$\Gamma, X : \tilde{S}\tilde{T}; C \vdash X\langle \tilde{e}, \tilde{k}\rangle \triangleright \Delta, \tilde{k} : \tilde{T}$ and $a \notin dom(\Gamma, X : \tilde{S}\tilde{T})$ By assumption
$\Gamma; C \vdash \tilde{e} : \tilde{S} \quad \Delta$ end only and $a \notin dom(\Gamma, X : \tilde{S}\tilde{T})$ By inversion
$\Gamma, a : \langle G\rangle; C \vdash \tilde{e} : \tilde{S} \quad \Delta$ end only By i.h.
$\Gamma, X : \tilde{S}\tilde{T}, a : \langle G\rangle; C \vdash X\langle \tilde{e}, \tilde{k}\rangle \triangleright \Delta, \tilde{k} : \tilde{T}$ By rule [TVar]

$\Gamma; C \vdash \mathsf{def}\ X(\tilde{y}, \tilde{y'}) = R\ \mathsf{in}\ R' \triangleright \Delta$ and $a \notin dom(\Gamma)$ By assumption
$\Gamma, X : \tilde{S}\tilde{T}, \tilde{y} : \tilde{S}; C \vdash R \triangleright \tilde{y'} : \tilde{T} \quad \Gamma, X : \tilde{S}\tilde{T}; C \vdash R' \triangleright \Delta$ and $a \notin dom(\Gamma)$ By inversion
$\Gamma, X : \tilde{S}\tilde{T}, \tilde{y} : \tilde{S}, a : \langle G\rangle; C \vdash R \triangleright \tilde{y'} : \tilde{T} \quad \Gamma, X : \tilde{S}\tilde{T}, a : \langle G\rangle; C \vdash R' \triangleright \Delta$ By i.h.
$\Gamma, a : \langle G\rangle; C \vdash \mathsf{def}\ X(\tilde{y}, \tilde{y'}) = R\ \mathsf{in}\ R' \triangleright \Delta$ By rule [TDef]

(2) is trivial by induction on the derivation of $\Gamma; C \vdash R \triangleright \Delta$. We present the most appealing cases, including those most difficult.

Cases [TFun], [TAppF], [TPRec], [TAppR] do not apply.

$\Gamma; C \vdash \bar{u}[\mathtt{p}_0, \mathtt{p}_1, p](y).R \triangleright \Delta$ and $\Delta' = \mathsf{end}\ only$ By assumption
$\Gamma; C \vdash u : \langle G\rangle \quad \emptyset; C \vdash G \blacktriangleright \mathsf{Type} \quad C \vdash \mathtt{p}_0, \mathtt{p}_1, p \quad C \vdash \mathsf{pid}(G) = \{\mathtt{p}_0, \mathtt{p}_1, p\}$
$\Gamma; C \vdash R \triangleright \Delta, y : \xi(G \upharpoonright \mathtt{p}_0)$ and $\Delta' = \mathsf{end}\ only$ By inversion
$\Gamma; C \vdash u : \langle G\rangle \quad \emptyset; C \vdash G \blacktriangleright \mathsf{Type} \quad C \vdash \mathtt{p}_0, \mathtt{p}_1, p \quad C \vdash \mathsf{pid}(G) = \{\mathtt{p}_0, \mathtt{p}_1, p\}$
$\Gamma; C \vdash R \triangleright \Delta, y : \xi(G \upharpoonright \mathtt{p}_0), \Delta'$ By induction
Note that $\Delta, y : \xi(G \upharpoonright \mathtt{p}_0), \Delta' = \Delta, \Delta', y : \xi(G \upharpoonright \mathtt{p}_0)$
$\Gamma; C \vdash \bar{u}[\mathtt{p}_0, \mathtt{p}_1, p](y).R \triangleright \Delta, \Delta'$ By rule [TMCast]

Case [TMAcc] is symmetric.

$\Gamma; C \vdash k!\langle \mathtt{p}, e\rangle; R \triangleright \Delta, k :!\langle \mathtt{p}, S\rangle; T$ and $\Delta' = \mathsf{end}\ only$ By assumption
$\Gamma; C \vdash e \triangleright S \quad C \vdash \mathtt{p} \quad \Gamma; C \vdash R \triangleright \Delta, k : T$ and $\Delta' = \mathsf{end}\ only$ By inversion
$\Gamma; C \vdash e \triangleright S \quad C \vdash \mathtt{p} \quad \Gamma; C \vdash R \triangleright \Delta, k : T, \Delta'$ By induction
Note that $\Delta, k : T, \Delta' = \Delta, \Delta', k : T$
$\Gamma; C \vdash k!\langle \mathtt{p}, e\rangle; R \triangleright \Delta, \Delta', k :!\langle \mathtt{p}, S\rangle; T$ By rule [TSend]
Note that $\Delta, \Delta', k :!\langle \mathtt{p}, S\rangle; T = \Delta, k :!\langle \mathtt{p}, S\rangle; T, \Delta'$

$\Gamma; C \vdash k?\langle \mathtt{p}, x\rangle; R \triangleright \Delta, k :?\langle \mathtt{p}, S\rangle; T$ and $\Delta' = \mathsf{end}\ only$ By assumption
$C \vdash \mathtt{p} \quad \Gamma, x : S; C \vdash R \triangleright \Delta, k : T$ and $\Delta' = \mathsf{end}\ only$ By inversion
$C \vdash \mathtt{p} \quad \Gamma, x : S; C \vdash R \triangleright \Delta, k : T, \Delta'$ By induction

Note that $\Delta, k : T, \Delta' = \Delta, \Delta', k : T$
$\Gamma; C \vdash k?\langle \mathtt{p}, x\rangle; R \triangleright \Delta, k :?\langle \mathtt{p}, S\rangle; T, \Delta'$ By rule [TRecv]
Note that $\Delta, k :?\langle \mathtt{p}, S\rangle; T, \Delta' = \Delta, \Delta', k :?\langle \mathtt{p}, S\rangle; T$.

Cases [TSel], [TBr] are symmetric.

$\Gamma; C \vdash R \mid S \triangleright \Delta, \Delta''$ and $\Delta' = \mathsf{end}$ *only* By assumption
$\Gamma; C \vdash R \triangleright \Delta \quad \Gamma; C \vdash S \triangleright \Delta''$ and $\Delta' = \mathsf{end}$ *only* By inversion
$\Gamma; C \vdash R \triangleright \Delta, \Delta' \quad \Gamma; C \vdash S \triangleright \Delta'', \Delta'$ By induction
$\Gamma \vdash P \mid Q \triangleright \Delta, \Delta', \Delta'', \Delta'$ By rule [Conc]
Note that $\Delta, \Delta', \Delta'', \Delta' = \Delta, \Delta'', \Delta'$.

Cases [TIf], [<:], [TInact], [TNRes], [TCRes] are symmetric.

$\Gamma, X : \tilde{S}\tilde{T}; C \vdash X\langle \tilde{e}, \tilde{k}\rangle \triangleright \Delta, \tilde{k} : \tilde{T}$ and $\Delta' = \mathsf{end}$ *only* By assumption
$\Gamma; C \vdash \tilde{e} : \tilde{S} \quad \Delta$ end only and $\Delta' = \mathsf{end}$ *only* By inversion
$\Gamma; C \vdash \tilde{e} : \tilde{S} \quad \Delta, \Delta'$ end only By induction
$\Gamma, X : \tilde{S}\tilde{T}; C \vdash X\langle \tilde{e}, \tilde{k}\rangle \triangleright \Delta, \Delta', \tilde{k} : \tilde{T}$ By rule [Var]
Note that $\Delta, \Delta', \tilde{k} : \tilde{T} = \Delta, \tilde{k} : \tilde{T}, \Delta'$.

$\Gamma; C \vdash \mathsf{def}\ X(\tilde{y}, \tilde{y'}) = R\ \mathsf{in}\ R' \triangleright \Delta$ and $\Delta' = \mathsf{end}$ *only* By assumption
$\Gamma, X : \tilde{S}\tilde{T}, \tilde{y} : \tilde{S}; C \vdash R \triangleright \tilde{y'} : \tilde{T} \quad \Gamma, X : \tilde{S}\tilde{T}; C \vdash R' \triangleright \Delta$ and $\Delta' = \mathsf{end}$ *only* By inversion
$\Gamma, X : \tilde{S}\tilde{T}, \tilde{y} : \tilde{S}; C \vdash R \triangleright \tilde{y'} : \tilde{T} \quad \Gamma, X : \tilde{S}\tilde{T}; C \vdash R' \triangleright \Delta, \Delta'$ By induction
$\Gamma; C \vdash \mathsf{def}\ X(\tilde{y}, \tilde{y'}) = R\ \mathsf{in}\ R' \triangleright \Delta, \Delta'$ By rule [Def]

(3) is trivial by induction on the derivation of $\Gamma; C \vdash E \triangleright \tau$. We present the most appealing cases, including the most difficult.

$\Gamma, u : \langle G'\rangle; C \vdash u : \langle G'\rangle$ and $X \notin dom(\Gamma)$ By assumption
$\Gamma, u : \langle G'\rangle, X : \tilde{S}\tilde{T}; C \vdash u : \langle G'\rangle$ By rule [TName]

$\Gamma; C \vdash \lambda n.E \triangleright \Pi n : \mathtt{T}.\tau$ and $X \notin dom(\Gamma)$ By assumption
$\Gamma; C, n : \mathtt{T} \vdash E \triangleright \tau$ and $X \notin dom(\Gamma)$ By inversion
$\Gamma, X : \tilde{S}\tilde{T}; C, n : \mathtt{T} \vdash E \triangleright \tau$ By i.h.
$\Gamma, X : \tilde{S}\tilde{T}; C \vdash \lambda n.E \triangleright \Pi n : \mathtt{T}.\tau$ By rule [TFun]

$\Gamma; C \vdash E\ \mathtt{t} \rhd \tau$ and $X \notin dom(\Gamma)$ By assumption
$\Gamma; C \vdash E \rhd \Pi n{:}\mathtt{T}.\tau \quad C \vdash \mathtt{t} \geq \mathsf{min}(\mathtt{T})$ and $X \notin dom(\Gamma)$ By inversion
$\Gamma, X{:}\tilde{S}\tilde{T}; C \vdash E \rhd \Pi n{:}\mathtt{T}.\tau \quad C \vdash \mathtt{t} \geq \mathsf{min}(\mathtt{T})$ By i.h.
$\Gamma, X{:}\tilde{S}\tilde{T}; C \vdash E\ \mathtt{t} \rhd \tau$ By rule [TAppF]

$\Gamma; C \vdash \mathbf{R}\ S\ \lambda i.\lambda X'.R \rhd \Pi i{:}\mathtt{I}.\Delta$ and $X \notin dom(\Gamma)$ By assumption
$\Gamma; C \vdash S \rhd \Delta\ 0 \quad \Gamma, X' : \Delta\ i; C, i : \mathtt{I} \vdash R \rhd \Delta\ i+1$ and $X \notin dom(\Gamma)$ By inversion
$\Gamma, X{:}\tilde{S}\tilde{T}; C \vdash S \rhd \Delta\ 0 \quad \Gamma, X' : \Delta\ i, X{:}\tilde{S}\tilde{T}; C, i : \mathtt{I} \vdash R \rhd \Delta\ i+1$ By i.h.
$\Gamma, X{:}\tilde{S}\tilde{T}; C \vdash \mathbf{R}\ S\ \lambda i.\lambda X'.R \rhd \Pi i{:}\mathtt{I}.\Delta$ By rule [TPRec]

$\Gamma; C \vdash R\ \mathtt{t} \rhd \Delta\ \mathtt{t}$ and $X \notin dom(\Gamma)$ By assumption
$C \vdash \mathtt{t} \quad \Gamma; C \vdash R \rhd \Pi i{:}\{i \mid i{\in}\mathsf{nat}, 0{\leq}i{\leq}\mathtt{t}{-}1\}.\Delta$ and $X \notin dom(\Gamma)$ By inversion
$C \vdash \mathtt{t} \quad \Gamma, X{:}\tilde{S}\tilde{T}; C \vdash R \rhd \Pi i{:}\{i \mid i{\in}\mathsf{nat}, 0{\leq}i{\leq}\mathtt{t}{-}1\}.\Delta$ By i.h.
$\Gamma, X{:}\tilde{S}\tilde{T}; C \vdash R\ \mathtt{t} \rhd \Delta\ \mathtt{t}$ By rule [TAppR]

$\Gamma; C \vdash \bar{u}[\mathtt{p}_0, \mathtt{p}_1, p](y).R \rhd \Delta$ and $X \notin dom(\Gamma)$ By assumption
$\Gamma; C \vdash u : \langle G' \rangle \quad \emptyset; C \vdash G \blacktriangleright \mathsf{Type} \quad C \vdash \mathtt{p}_0, \mathtt{p}_1, p \quad C \vdash \mathsf{pid}(G') = \{\mathtt{p}_0, \mathtt{p}_1, p\}$
$\Gamma; C \vdash R \rhd \Delta, y : \xi(G' \upharpoonright \mathtt{p}_0)$ and $X \notin dom(\Gamma)$ By inversion
$\Gamma, X{:}\tilde{S}\tilde{T}; C \vdash u : \langle G' \rangle \quad \emptyset; C \vdash G \blacktriangleright \mathsf{Type} \quad C \vdash \mathtt{p}_0, \mathtt{p}_1, p \quad C \vdash \mathsf{pid}(G') = \{\mathtt{p}_0, \mathtt{p}_1, p\}$
$\Gamma, X{:}\tilde{S}\tilde{T}; C \vdash R \rhd \Delta, y : \xi(G' \upharpoonright \mathtt{p}_0)$ By i.h.
$\Gamma, X{:}\tilde{S}\tilde{T}; C \vdash \bar{u}[\mathtt{p}_0, \mathtt{p}_1, p](y).R \rhd \Delta$ By rule [TMCast]

Case [TMAcc] is symmetric.

$\Gamma; C \vdash k!\langle \mathtt{p}, e \rangle; R \rhd \Delta, k :!\langle \mathtt{p}, S \rangle; T$ and $X \notin dom(\Gamma)$ By assumption
$\Gamma; C \vdash e \rhd S \quad C \vdash \mathtt{p} \quad \Gamma; C \vdash R \rhd \Delta, k : T$ and $X \notin dom(\Gamma)$ By inversion
$\Gamma, X{:}\tilde{S}\tilde{T}; C \vdash e \rhd S \quad C \vdash \mathtt{p} \quad \Gamma, X{:}\tilde{S}\tilde{T}; C \vdash R \rhd \Delta, k : T$ By i.h.
$\Gamma, X{:}\tilde{S}\tilde{T}; C \vdash k!\langle \mathtt{p}, e \rangle; R \rhd \Delta, k :!\langle \mathtt{p}, S \rangle; T$ By rule [TSend]

Cases [TRcv], [TSel], [TBr] are symmetric.

$\Gamma; C \vdash (\nu b)R \rhd \Delta$ and $X \notin dom(\Gamma)$ By assumption
$\Gamma, b : \langle G \rangle; C \vdash R \rhd \Delta$ and $X \notin dom(\Gamma)$ By inversion
$\Gamma, b : \langle G \rangle, X{:}\tilde{S}\tilde{T}; C \vdash R \rhd \Delta$ By i.h.

$\Gamma, X:\tilde{S}\tilde{T}; C \vdash (\nu b)R \triangleright \Delta$ By rule [TNu]

$\Gamma; C \vdash R \mid S \triangleright \Delta, \Delta'$ and $X \notin dom(\Gamma)$ By assumption
$\Gamma; C \vdash R \triangleright \Delta \quad \Gamma; C \vdash S \triangleright \Delta'$ and $X \notin dom(\Gamma)$ By inversion
$\Gamma, X:\tilde{S}\tilde{T}; C \vdash R \triangleright \Delta \quad \Gamma, X:\tilde{S}\tilde{T}; C \vdash S \triangleright \Delta'$ By i.h.
$\Gamma, X:\tilde{S}\tilde{T}; C \vdash R \mid S \triangleright \Delta, \Delta'$ By rule [TConc]

Cases [TVar], [TInact], [TIf], [<:] are trivial.

$\Gamma; C \vdash (\nu s)R \triangleright \Delta$ and $X \notin dom(\Gamma)$ By assumption
$\Gamma; C \vdash R \triangleright \Delta, s[\hat{\mathrm{p}}_1] : T_1, ..., s[\hat{\mathrm{p}}_n] : T_\mathrm{n}$ and $X \notin dom(\Gamma)$ By inversion
$\Gamma, X:\tilde{S}\tilde{T}'; C \vdash R \triangleright \Delta, s[\hat{\mathrm{p}}_1] : T_1, ..., s[\hat{\mathrm{p}}_n] : T_\mathrm{n}$ By i.h.
$\Gamma, X:\tilde{S}\tilde{T}'; C \vdash (\nu s)R \triangleright \Delta$ By rule [TCres]

$\Gamma, X':\tilde{S}\tilde{T}; C \vdash X'\langle \tilde{e}, \tilde{k}\rangle \triangleright \Delta, \tilde{k} : \tilde{T}$ and $X \notin dom(\Gamma)$ By assumption
$\Gamma; C \vdash \tilde{e}:\tilde{S} \qquad \Delta$ end only and $X \notin dom(\Gamma)$ By inversion
$\Gamma, X:\tilde{S}'\tilde{T}'; C \vdash \tilde{e}:\tilde{S} \qquad \Delta$ end only By i.h.
$\Gamma, X':\tilde{S}\tilde{T}, X:\tilde{S}'\tilde{T}'; C \vdash X'\langle \tilde{e}, \tilde{k}\rangle \triangleright \Delta, \tilde{k} : \tilde{T}$ By rule [TVar]

$\Gamma; C \vdash \mathsf{def}\ X'(\tilde{y}, \tilde{y'}) = R\ \mathsf{in}\ R' \triangleright \Delta$ and $X \notin dom(\Gamma)$ By assumption
$\Gamma, X':\tilde{S}\tilde{T}, \tilde{y} : \tilde{S}; C \vdash R \triangleright \tilde{y'} : \tilde{T} \quad \Gamma, X':\tilde{S}\tilde{T}; C \vdash R' \triangleright \Delta$ and $X \notin dom(\Gamma)$ By inversion
$\Gamma, X':\tilde{S}\tilde{T}, \tilde{y} : \tilde{S}, X:\tilde{S}'\tilde{T}'; C \vdash R \triangleright \tilde{y'} : \tilde{T} \quad \Gamma, X':\tilde{S}\tilde{T}, X:\tilde{S}'\tilde{T}'; C \vdash R' \triangleright \Delta$ By i.h.
$\Gamma, X:\tilde{S}'\tilde{T}'; C \vdash \mathsf{def}\ X'(\tilde{y}, \tilde{y'}) = R\ \mathsf{in}\ R' \triangleright \Delta$ By rule [TDef]

□

Lemma 3.4.53 (Strengthening Lemma)**.** 1. Let $\Gamma; C \vdash E \triangleright \tau$. If $a \notin fn(E)$, then $\Gamma \setminus a; C \vdash E \triangleright \tau$.

2. Let $\Gamma; C \vdash R \triangleright \Delta$. If $s \notin fc(R)$, then $\Gamma; C \vdash R \triangleright \Delta \setminus s$.

3. Let $\Gamma; C \vdash E \triangleright \tau$. If $X \notin \mathrm{fpv}(E)$, then $\Gamma \setminus X; C \vdash E \triangleright \tau$.

Proof. (1) is trivial by induction on the derivation of $\Gamma; C \vdash E \triangleright \tau$. We present the most appealing cases, including the most difficult.

$\Gamma, u{:}\langle G'\rangle; C \vdash u{:}\ \langle G'\rangle$ and $a \notin fn(u)$ — By assumption
$\Gamma \setminus a, u{:}\langle G'\rangle; C \vdash u{:}\ \langle G'\rangle$ — By rule [TName]

$\Gamma; C \vdash \lambda n.E \rhd \Pi n{:}\mathtt{T}.\tau$ and $a \notin fn(\lambda n.E)$ — By assumption
$\Gamma; C, n : \mathtt{T} \vdash E \rhd \tau$ and $a \notin fn(E)$ — By inversion and Definition 3.4.1
$\Gamma \setminus a; C, n : \mathtt{T} \vdash E \rhd \tau$ — By i.h.
$\Gamma \setminus a; C \vdash \lambda n.E \rhd \Pi n{:}\mathtt{T}.\tau$ — By rule [TFun]

$\Gamma; C \vdash E\ \mathtt{t} \rhd \tau$ and $a \notin fn(E\ \mathtt{t})$ — By assumption
$\Gamma; C \vdash E \rhd \Pi n{:}\mathtt{T}.\tau \quad C \vdash \mathtt{t} \geq \mathsf{min}(\mathtt{T})$ and $a \notin fn(E)$ — By inversion and Definition 3.4.1
$\Gamma \setminus a; C \vdash E \rhd \Pi n{:}\mathtt{T}.\tau \quad C \vdash \mathtt{t} \geq \mathsf{min}(\mathtt{T})$ — By i.h.
$\Gamma \setminus a; C \vdash E\ \mathtt{t} \rhd \tau$ — By rule [TAppF]

$\Gamma; C \vdash \mathbf{R}\ S\ \lambda i.\lambda X.R \rhd \Pi i{:}\mathtt{I}.\Delta$ and $a \notin fn(\mathbf{R}\ S\ \lambda i.\lambda X.R)$ — By assumption
$\Gamma; C \vdash S \rhd \Delta\ 0 \quad \Gamma, X : \Delta\ i; C, i : \mathtt{I} \vdash R \rhd \Delta\ i + 1$ and $a \notin fn(S)$
and $a \notin fn(R)$ — By inversion and Definition 3.4.1
$\Gamma \setminus a; C \vdash S \rhd \Delta\ 0 \quad \Gamma, X : \Delta\ i \setminus a; C, i : \mathtt{I} \vdash R \rhd \Delta\ i + 1$ — By i.h.
$\Gamma \setminus a; C \vdash \mathbf{R}\ S\ \lambda i.\lambda X.R \rhd \Pi i{:}\mathtt{I}.\Delta$ — By rule [TPRec]

$\Gamma; C \vdash R\ \mathtt{t} \rhd \Delta\ \mathtt{t}$ and $a \notin fn(R\ \mathtt{t})$ — By assumption
$C \vdash \mathtt{t} \quad \Gamma; C \vdash R \rhd \Pi i{:}\{i \mid i{\in}\mathsf{nat}, 0{\leq}i{\leq}\mathtt{t}{-}1\}.\Delta$
and $a \notin fn(R)$ — By inversion and Definition 3.4.1
$C \vdash \mathtt{t} \quad \Gamma \setminus a; C \vdash R \rhd \Pi i{:}\{i \mid i{\in}\mathsf{nat}, 0{\leq}i{\leq}\mathtt{t}{-}1\}.\Delta$ — By i.h.
$\Gamma \setminus a; C \vdash R\ \mathtt{t} \rhd \Delta\ \mathtt{t}$ — By rule [TAppR]

$\Gamma; C \vdash \bar{u}[\mathtt{p}_0, \mathtt{p}_1, p](y).R \rhd \Delta$ and $a \notin fn(\bar{u}[\mathtt{p}_0, \mathtt{p}_1, p](y).R)$ — By assumption
$\Gamma; C \vdash u : \langle G'\rangle \quad \emptyset; C \vdash G \blacktriangleright \mathsf{Type} \quad C \vdash \mathtt{p}_0, \mathtt{p}_1, p \quad C \vdash \mathsf{pid}(G') = \{\mathtt{p}_0, \mathtt{p}_1, p\}$
$\Gamma; C \vdash R \rhd \Delta, y : \xi(G' \upharpoonright \mathtt{p}_0)$ and $a \notin fn(u) \bigcup fn(R)$ — By inversion and Definition 3.4.1
$\Gamma \setminus a; C \vdash u : \langle G'\rangle \quad \emptyset; C \vdash G \blacktriangleright \mathsf{Type} \quad C \vdash \mathtt{p}_0, \mathtt{p}_1, p \quad C \vdash \mathsf{pid}(G') = \{\mathtt{p}_0, \mathtt{p}_1, p\}$
$\Gamma \setminus a; C \vdash R \rhd \Delta, y : \xi(G' \upharpoonright \mathtt{p}_0)$ — By i.h.
$\Gamma \setminus a; C \vdash \bar{u}[\mathtt{p}_0, \mathtt{p}_1, p](y).R \rhd \Delta$ — By rule [TMCast]

Case [TMAcc] is symmetric.

$\Gamma; C \vdash k!\langle \mathtt{p}, e\rangle; R \rhd \Delta, k :!\langle \mathtt{p}, S\rangle; T$ and $a \notin fn(k!\langle \mathtt{p}, e\rangle; R)$ — By assumption

$\Gamma; C \vdash e \triangleright S \quad C \vdash \mathtt{p} \quad \Gamma; C \vdash R \triangleright \Delta, k : T$
and $a \notin fn(\tilde{e}) \bigcup fn(R)$ By inversion and Definition 3.4.1
$\Gamma \setminus a; C \vdash e \triangleright S \quad C \vdash \mathtt{p} \quad \Gamma \setminus a; C \vdash R \triangleright \Delta, k : T$ By i.h.
$\Gamma \setminus a; C \vdash k!\langle \mathtt{p}, e \rangle; R \triangleright \Delta, k :!\langle \mathtt{p}, S \rangle; T$ By rule [TSend]

Cases [TRcv], [TSel], [TBr] are symmetric.

$\Gamma; C \vdash (\nu b)R \triangleright \Delta$ and $a \notin fn((\nu b)R)$ By assumption
$\Gamma, b : U; C \vdash R \triangleright \Delta$ and $a \notin fn(R) \setminus \{b\}$ By inversion and Definition 3.4.1
$\Gamma, b : U \setminus a; C \vdash R \triangleright \Delta$ By i.h.
$\Gamma \setminus a; C \vdash (\nu b)R \triangleright \Delta$ By rule [TNu]

$\Gamma; C \vdash R \mid S \triangleright \Delta, \Delta'$ and $a \notin fn(R \mid S)$ By assumption
$\Gamma; C \vdash R \triangleright \Delta \quad \Gamma; C \vdash S \triangleright \Delta'$ and $a \notin fn(R) \bigcup fn(S)$ By inversion and Definition 3.4.1
$\Gamma \setminus a; C \vdash R \triangleright \Delta \quad \Gamma \setminus a; C \vdash S \triangleright \Delta'$ By i.h.
$\Gamma \setminus a; C \vdash R \mid S \triangleright \Delta, \Delta'$ By rule [TConc]

Cases [TVar], [TInact], [TIf], [<:] are trivial.

$\Gamma; C \vdash (\nu s)R \triangleright \Delta$ and $a \notin fn((\nu s)R)$ By assumption
$\Gamma; C \vdash R \triangleright \Delta, s[\hat{\mathtt{p}}_1] : T_1, ..., s[\hat{\mathtt{p}}_n] : T_\mathrm{n}$ and $a \notin fn(R)$ By inversion and Definition 3.4.1
$\Gamma \setminus a; C \vdash R \triangleright \Delta, s[\hat{\mathtt{p}}_1] : T_1, ..., s[\hat{\mathtt{p}}_n] : T_\mathrm{n}$ By i.h.
$\Gamma \setminus a; C \vdash (\nu s)R \triangleright \Delta$ By rule [TCres]

$\Gamma, X : \tilde{S}\tilde{T}; C \vdash X\langle \tilde{e}, \tilde{k} \rangle \triangleright \Delta, \tilde{k} : \tilde{T}$ and $a \notin fn(X\langle \tilde{e}, \tilde{k} \rangle)$ By assumption
$\Gamma; C \vdash \tilde{e} : \tilde{S} \quad \Delta$ end only and $a \notin fn(\tilde{e})$ By inversion and Definition 3.4.1
$\Gamma \setminus a; C \vdash \tilde{e} : \tilde{S} \quad \Delta$ end only By i.h.
$\Gamma, X : \tilde{S}\tilde{T} \setminus a; C \vdash X\langle \tilde{e}, \tilde{k} \rangle \triangleright \Delta, \tilde{k} : \tilde{T}$ By rule [TVar]

$\Gamma; C \vdash \mathsf{def}\ X(\tilde{y}, \tilde{y'}) = R\ \mathsf{in}\ R' \triangleright \Delta$ and $a \notin fn(\mathsf{def}\ X(\tilde{y}, \tilde{y'}) = R\ \mathsf{in}\ R')$ By assumption
$\Gamma, X : \tilde{S}\tilde{T}, \tilde{y} : \tilde{S}; C \vdash R \triangleright \tilde{y'} : \tilde{T} \quad \Gamma, X : \tilde{S}\tilde{T}; C \vdash R' \triangleright \Delta$

and $a \notin fn(R') \bigcup fn(R)$ By inversion
$\Gamma, X : \tilde{S}\tilde{T}, \tilde{y} : \tilde{S} \setminus a; C \vdash R \triangleright \tilde{y'} : \tilde{T} \quad \Gamma, X : \tilde{S}\tilde{T} \setminus a; C \vdash R' \triangleright \Delta$ By i.h.
$\Gamma \setminus a; C \vdash \mathsf{def}\ X(\tilde{y}, \tilde{y'}) = R\ \mathsf{in}\ R' \triangleright \Delta$ By rule [TDef]

(2) is trivial by induction on the derivation of $\Gamma; C \vdash R \triangleright \Delta$. We present the most appealing cases, including those most difficult.

$\Gamma, u : \langle G \rangle; C \vdash u : \langle G \rangle$ and $s \notin fc(u)$ By assumption
$\Gamma, u : \langle G \rangle; C \vdash u : \langle G \rangle \setminus s$ By rule [TName]
Note that $u : \langle G \rangle \setminus s = u : \langle G \rangle$

Cases [TFun], [TAppF], [TPRec], [TAppR] do not apply.

$\Gamma; C \vdash \bar{u}[\mathtt{p}_0, \mathtt{p}_1, p](y).R \triangleright \Delta$ and $s \notin fc(\bar{u}[\mathtt{p}_0, \mathtt{p}_1, p](y).R)$ By assumption
$\Gamma; C \vdash u : \langle G \rangle \quad \emptyset; C \vdash G \blacktriangleright \mathsf{Type} \quad C \vdash \mathtt{p}_0, \mathtt{p}_1, p \quad C \vdash \mathsf{pid}(G) = \{\mathtt{p}_0, \mathtt{p}_1, p\}$
$\Gamma; C \vdash R \triangleright \Delta, y : \xi(G \upharpoonright \mathtt{p}_0)$ and $s \notin fc(u) \bigcup fc(R)$ By inversion and Definition 3.4.2
$\Gamma; C \vdash u : \langle G \rangle \setminus s \quad \emptyset; C \vdash G \blacktriangleright \mathsf{Type} \quad C \vdash \mathtt{p}_0, \mathtt{p}_1, p \quad C \vdash \mathsf{pid}(G) = \{\mathtt{p}_0, \mathtt{p}_1, p\}$
$\Gamma; C \vdash R \triangleright \Delta, y : \xi(G \upharpoonright \mathtt{p}_0) \setminus s$ By induction
Note that $\Delta, y : \xi(G \upharpoonright \mathtt{p}_0) \setminus s = \Delta \setminus s, y : \xi(G \upharpoonright \mathtt{p}_0)$
$\Gamma; C \vdash \bar{u}[\mathtt{p}_0, \mathtt{p}_1, p](y).R \triangleright \Delta \setminus s$ By rule [TMCast]

Case [TMAcc] is symmetric.

$\Gamma; C \vdash k!\langle \mathtt{p}, e \rangle; R \triangleright \Delta, k : !\langle \mathtt{p}, S \rangle; T$ and $s \notin fc(k!\langle \mathtt{p}, e \rangle; R)$ By assumption
$\Gamma; C \vdash e \triangleright S \quad C \vdash \mathtt{p} \quad \Gamma; C \vdash R \triangleright \Delta, k : T$
and $s \notin fc(k) \bigcup fn(R)$ By inversion and Definition 3.4.2
$\Gamma; C \vdash e \triangleright S \quad C \vdash \mathtt{p} \quad \Gamma; C \vdash R \triangleright \Delta, k : T \setminus s$ By induction
Note that $\Delta, k : T \setminus s = \Delta \setminus s, k : T$
$\Gamma; C \vdash k!\langle \mathtt{p}, e \rangle; R \triangleright \Delta \setminus s, k : !\langle \mathtt{p}, S \rangle; T$ By rule [TSend]
Note that $\Delta \setminus s, k : !\langle \mathtt{p}, S \rangle; T = \Delta, k : !\langle \mathtt{p}, S \rangle; T \setminus s$

Cases [TRcv], [TSel], [TBr] are symmetric.

$\Gamma; C \vdash R \mid S \triangleright \Delta, \Delta''$ and $s \notin fc(R \mid S)$ By assumption
$\Gamma; C \vdash R \triangleright \Delta \quad \Gamma; C \vdash S \triangleright \Delta''$ and $s \notin fc(R) \bigcup fc(S)$ By inversion and Definition 3.4.2
$\Gamma; C \vdash R \triangleright \Delta \setminus s \quad \Gamma; C \vdash S \triangleright \Delta'' \setminus s$ By induction
$\Gamma \vdash P \mid Q \triangleright \Delta \setminus s, \Delta'' \setminus s$ By rule [Conc]
Note that $\Delta \setminus s, \Delta'' \setminus s = \Delta, \Delta'' \setminus s$.

Cases [TIf], [<:], [TInact] are symmetric.

$\Gamma; C \vdash (\nu s')R \triangleright \Delta$ and $s \notin (\nu s')R$ By assumption
$\Gamma; C \vdash R \triangleright \Delta, s'[\hat{\mathtt{p}}_1] : T_1, ..., s'[\hat{\mathtt{p}}_n] : T_{\mathrm{n}}$ and $s \notin fc(R) \setminus s'$ By inversion and Definition 3.4.2
$\Gamma; C \vdash R \triangleright \Delta, s'[\hat{\mathtt{p}}_1] : T_1, ..., s'[\hat{\mathtt{p}}_n] : T_{\mathrm{n}} \setminus s$ By i.h.
Note that $\Delta, s'[\hat{\mathtt{p}}_1] : T_1, ..., s'[\hat{\mathtt{p}}_n] : T_{\mathrm{n}} \setminus s = \Delta \setminus s, s'[\hat{\mathtt{p}}_1] : T_1, ..., s'[\hat{\mathtt{p}}_n] : T_{\mathrm{n}}$
$\Gamma; C \vdash (\nu s)R \triangleright \Delta \setminus s$ By rule [TCres]

$\Gamma, X : \tilde{S}\tilde{T}; C \vdash X\langle \tilde{e}, \tilde{k} \rangle \triangleright \Delta, \tilde{k} : \tilde{T}$ and $s \notin fc(X\langle \tilde{e}, \tilde{k} \rangle)$ By assumption
$\Gamma; C \vdash \tilde{e} : \tilde{S} \quad \Delta$ end only and $s \notin fc(\tilde{k})$ By inversion and Definition 3.4.2
$\Gamma; C \vdash \tilde{e} : \tilde{S} \quad \Delta \setminus s$ end only By induction
$\Gamma, X : \tilde{S}\tilde{T}; C \vdash X\langle \tilde{e}, \tilde{k} \rangle \triangleright \Delta \setminus s, \tilde{k} : \tilde{T}$ By rule [Var]
Note that $\Delta \setminus s, \tilde{k} : \tilde{T} = \Delta, \tilde{k} : \tilde{T} \setminus s$.

$\Gamma; C \vdash \mathsf{def}\ X(\tilde{y}, \tilde{y'}) = R\ \mathsf{in}\ R' \triangleright \Delta$ and $s \notin fc(\mathsf{def}\ X(\tilde{y}, \tilde{y'}) = R\ \mathsf{in}\ R')$ By assumption
$\Gamma, X : \tilde{S}\tilde{T}, \tilde{y} : \tilde{S}; C \vdash R \triangleright \tilde{y'} : \tilde{T} \quad \Gamma, X : \tilde{S}\tilde{T}; C \vdash R' \triangleright \Delta$ and $s \notin fc(R) \bigcup fc(R')$ By inversion
$\Gamma, X : \tilde{S}\tilde{T}, \tilde{y} : \tilde{S}; C \vdash R \triangleright \tilde{y'} : \tilde{T} \quad \Gamma, X : \tilde{S}\tilde{T}; C \vdash R' \triangleright \Delta \setminus s'$ By induction
$\Gamma; C \vdash \mathsf{def}\ X(\tilde{y}, \tilde{y'}) = R\ \mathsf{in}\ R' \triangleright \Delta \setminus s$ By rule [Def]

(3) is trivial by induction on the derivation of $\Gamma; C \vdash E \triangleright \tau$. We present the most appealing cases, including the most difficult.

$\Gamma; C \vdash \lambda n.E \triangleright \Pi n : \mathtt{T}.\tau$ and $X \notin \mathrm{fpv}(\lambda n.E)$ By assumption
$\Gamma; C, n : \mathtt{T} \vdash E \triangleright \tau$ and $X \notin \mathrm{fpv}(E)$ By inversion and Definition 3.4.5
$\Gamma \setminus X; C, n : \mathtt{T} \vdash E \triangleright \tau$ By i.h.
$\Gamma \setminus X; C \vdash \lambda n.E \triangleright \Pi n : \mathtt{T}.\tau$ By rule [TFun]

$\Gamma; C \vdash E\ \mathtt{t} \rhd \tau$ and $X \notin \mathrm{fpv}(E\ \mathtt{t})$ By assumption
$\Gamma; C \vdash E \rhd \Pi n{:}\mathtt{T}.\tau \quad C \vdash \mathtt{t} \geq \mathsf{min}(\mathtt{T})$ and $X \notin \mathrm{fpv}(E)$ By inversion and Definition 3.4.5
$\Gamma \setminus X; C \vdash E \rhd \Pi n{:}\mathtt{T}.\tau \quad C \vdash \mathtt{t} \geq \mathsf{min}(\mathtt{T})$ By i.h.
$\Gamma \setminus X; C \vdash E\ \mathtt{t} \rhd \tau$ By rule [TAppF]

$\Gamma; C \vdash \mathbf{R}\ S\ \lambda i.\lambda X'.R \rhd \Pi i{:}\mathtt{I}.\Delta$ and $X \notin \mathrm{fpv}(\mathbf{R}\ S\ \lambda i.\lambda X'.R)$ By assumption
$\Gamma; C \vdash S \rhd \Delta\ 0 \quad \Gamma, X' : \Delta\ i; C, i : \mathtt{I} \vdash R \rhd \Delta\ i+1$
and $X \notin \mathrm{fpv}(S) \bigcup (\mathrm{fpv}(R) \setminus \{X\})$ By inversion and Definition 3.4.5
$\Gamma \setminus X; C \vdash S \rhd \Delta\ 0 \quad \Gamma, X' : \Delta\ i \setminus X; C, i : \mathtt{I} \vdash R \rhd \Delta\ i+1$ By i.h.
$\Gamma \setminus X; C \vdash \mathbf{R}\ S\ \lambda i.\lambda X'.R \rhd \Pi i{:}\mathtt{I}.\Delta$ By rule [TPRec]

$\Gamma; C \vdash R\ \mathtt{t} \rhd \Delta\ \mathtt{t}$ and $X \notin \mathrm{fpv}(R\ \mathtt{t})$ By assumption
$C \vdash \mathtt{t} \quad \Gamma; C \vdash R \rhd \Pi i{:}\{i \mid i{\in}\mathsf{nat}, 0{\leq}i{\leq}\mathtt{t}{-}1\}.\Delta$
and $X \notin \mathrm{fpv}(R)$ By inversion and Definition 3.4.5
$C \vdash \mathtt{t} \quad \Gamma \setminus X; C \vdash R \rhd \Pi i{:}\{i \mid i{\in}\mathsf{nat}, 0{\leq}i{\leq}\mathtt{t}{-}1\}.\Delta$ By i.h.
$\Gamma \setminus X; C \vdash R\ \mathtt{t} \rhd \Delta\ \mathtt{t}$ By rule [TAppR]

$\Gamma; C \vdash \bar{u}[\mathtt{p}_0, \mathtt{p}_1, p](y).R \rhd \Delta$ and $X \notin \mathrm{fpv}(\bar{u}[\mathtt{p}_0, \mathtt{p}_1, p](y).R)$ By assumption
$\Gamma; C \vdash u : \langle G' \rangle \quad \emptyset; C \vdash G \blacktriangleright \mathsf{Type} \quad C \vdash \mathtt{p}_0, \mathtt{p}_1, p \quad C \vdash \mathsf{pid}(G') = \{\mathtt{p}_0, \mathtt{p}_1, p\}$
$\Gamma; C \vdash R \rhd \Delta, y : \xi(G' \upharpoonright \mathtt{p}_0)$ and $X \notin \mathrm{fpv}(R)$ By inversion and Definition 3.4.5
$\Gamma \setminus X; C \vdash u : \langle G' \rangle \quad \emptyset; C \vdash G \blacktriangleright \mathsf{Type} \quad C \vdash \mathtt{p}_0, \mathtt{p}_1, p \quad C \vdash \mathsf{pid}(G') = \{\mathtt{p}_0, \mathtt{p}_1, p\}$
$\Gamma \setminus X; C \vdash R \rhd \Delta, y : \xi(G' \upharpoonright \mathtt{p}_0)$ By i.h.
$\Gamma \setminus X; C \vdash \bar{u}[\mathtt{p}_0, \mathtt{p}_1, p](y).R \rhd \Delta$ By rule [TMCast]

Case [TMAcc] is symmetric.

$\Gamma; C \vdash k!\langle \mathtt{p}, e \rangle; R \rhd \Delta, k : !\langle \mathtt{p}, S \rangle; T$ and $X \notin \mathrm{fpv}(k!\langle \mathtt{p}, e \rangle; R)$ By assumption
$\Gamma; C \vdash e \rhd S \quad C \vdash \mathtt{p} \quad \Gamma; C \vdash R \rhd \Delta, k : T$
and $X \notin \mathrm{fpv}(R)$ By inversion and Definition 3.4.5
$\Gamma \setminus X; C \vdash e \rhd S \quad \Gamma \setminus X; C \vdash R \rhd \Delta, k : T$ By definition of e and i.h.
$\Gamma \setminus X; C \vdash k!\langle \mathtt{p}, e \rangle; R \rhd \Delta, k : !\langle \mathtt{p}, S \rangle; T$ By rule [TSend]

Cases [TRcv], [TSel], [TBr] are symmetric.

$\Gamma; C \vdash (\nu b)R \triangleright \Delta$ and $X \notin \text{fpv}((\nu b)R)$ By assumption
$\Gamma, b : \langle G \rangle; C \vdash R \triangleright \Delta$ and $X \notin \text{fpv}(R)$ By inversion and Definition 3.4.5
$\Gamma, b : \langle G \rangle \setminus X; C \vdash R \triangleright \Delta$ By i.h.
$\Gamma \setminus X; C \vdash (\nu b)R \triangleright \Delta$ By rule [TNu]

$\Gamma; C \vdash R \mid S \triangleright \Delta, \Delta'$ and $X \notin \text{fpv}(R \mid S)$ By assumption
$\Gamma; C \vdash R \triangleright \Delta \quad \Gamma; C \vdash S \triangleright \Delta'$ and $X \notin \text{fpv}(R) \bigcup \text{fpv}(S)$ By inversion and Definition 3.4.5
$\Gamma \setminus X; C \vdash R \triangleright \Delta \quad \Gamma \setminus X; C \vdash S \triangleright \Delta'$ By i.h.
$\Gamma \setminus X; C \vdash R \mid S \triangleright \Delta, \Delta'$ By rule [TConc]

Cases [TVar], [TInact], [TIf], [<:], [TNu] are trivial.

$\Gamma, X' : \tilde{S}\tilde{T}; C \vdash X'\langle \tilde{e}, \tilde{k} \rangle \triangleright \Delta, \tilde{k} : \tilde{T}$ and $X \notin \text{fpv}(X'\langle \tilde{e}, \tilde{k} \rangle)$ By assumption
$\Gamma; C \vdash \tilde{e} : \tilde{S} \qquad \Delta$ end only and $X \notin \{X'\}$ By inversion and Definition 3.4.5
$\Gamma \setminus X; C \vdash \tilde{e} : \tilde{S} \qquad \Delta$ end only By i.h.
$\Gamma, X' : \tilde{S}\tilde{T} \setminus X; C \vdash X'\langle \tilde{e}, \tilde{k} \rangle \triangleright \Delta, \tilde{k} : \tilde{T}$ By rule [TVar]

$\Gamma; C \vdash \mathsf{def}\ X'(\tilde{y}, \tilde{y'}) = R\ \mathsf{in}\ R' \triangleright \Delta$ and $X \notin \text{fpv}(\mathsf{def}\ X'(\tilde{y}, \tilde{y'}) = R\ \mathsf{in}\ R')$ By assumption
$\Gamma, X' : \tilde{S}\tilde{T}, \tilde{y} : \tilde{S}; C \vdash R \triangleright \tilde{y'} : \tilde{T} \quad \Gamma, X' : \tilde{S}\tilde{T}; C \vdash R' \triangleright \Delta$
and $X \notin \text{fpv}(R') \bigcup \text{fpv}(R) \setminus \{X'\}$ By inversion
$\Gamma, X' : \tilde{S}\tilde{T}, \tilde{y} : \tilde{S} \setminus X; C \vdash R \triangleright \tilde{y'} : \tilde{T} \quad \Gamma, X' : \tilde{S}\tilde{T} \setminus X; C \vdash R' \triangleright \Delta$ By i.h.
$\Gamma \setminus X; C \vdash \mathsf{def}\ X'(\tilde{y}, \tilde{y'}) = R\ \mathsf{in}\ R' \triangleright \Delta$ By rule [TDef]

□

Theorem 3.4.54 (subject congruence)**.** $\Gamma; C \vdash R \triangleright \Delta$ and $R \equiv R'$ imply $\Gamma; C \vdash R' \triangleright \Delta$.

Proof. By rule induction on the derivation of $\Gamma; C \vdash R \triangleright \Delta$ when assuming that $R \equiv R'$ and $\Gamma; C \vdash R \triangleright \Delta$. For each structural congruence axiom, we consider each session type system rule that can generate $\Gamma; C \vdash R \triangleright \Delta$.

Case: $R \mid \mathbf{0} \equiv R$

$\Gamma; C \vdash R \mid \mathbf{0} \triangleright \Delta$ By assumption

$\Gamma; C \vdash R \triangleright \Delta_1$ and $\Gamma; C \vdash \mathbf{0} \triangleright \Delta_2$ where $\Delta = \Delta_1, \Delta_2$ — By inversion
Δ_2 is only end and for Δ_2 such that $dom(\Delta_1) \cap dom(\Delta_2) = \emptyset$ — By inversion
then $\Gamma; C \vdash R \triangleright \Delta_1, \Delta_2$ — By weakening
and $\Delta = \Delta_1, \Delta_2$

$\Gamma; C \vdash R \triangleright \Delta$ — By assumption
$\Gamma; C \vdash \mathbf{0} \triangleright \Delta'$ where Δ' is only end s.t. $dom(\Delta) \cap dom(\Delta') = \emptyset$ — By rule $\lfloor$TInact$\rfloor$
$\Gamma; C \vdash R \mid \mathbf{0} \triangleright \Delta, \Delta'$ — By rule $\lfloor$TConc$\rfloor$
$\Gamma; C \vdash R \mid \mathbf{0} \triangleright \Delta$ — By Lemma 3.4.53(2)

Case: $R \mid S \equiv S \mid R$

$\Gamma; C \vdash R \mid S \triangleright \Delta$ — By assumption
$\Gamma; C \vdash R \triangleright \Delta_1$ and $\Gamma; C \vdash S \triangleright \Delta_2$ where $\Delta = \Delta_1, \Delta_2$ — By inversion
$\Gamma; C \vdash S \mid R \triangleright \Delta$ — By rule $\lfloor$TConc$\rfloor$

The other case is symmetric to the above one.

Case: $(R \mid S) \mid T \equiv R \mid (S \mid T)$

$\Gamma; C \vdash (R \mid S) \mid T \triangleright \Delta$ — By assumption
$\Gamma; C \vdash R \triangleright \Delta_1$, $\Gamma; C \vdash S \triangleright \Delta_2$ and $\Gamma; C \vdash T \triangleright \Delta_3$ where $\Delta = \Delta_1, \Delta_2, \Delta_3$ — By inversion
$\Gamma; C \vdash R \mid (S \mid T) \triangleright \Delta$ — By rule $\lfloor$TConc$\rfloor$

The other case is symmetric to the above one.

Case: $(\nu w)R \mid S \equiv (\nu w)(R \mid S) \quad$ if $w \notin fn(S) \cup fc(S)$

$\Gamma; C \vdash (\nu w)R \mid S \triangleright \Delta$ — By assumption
$\Gamma; C \vdash (\nu w)R \triangleright \Delta_1$ and $\Gamma; C \vdash S \triangleright \Delta_2$, $\Delta = \Delta_1, \Delta_2$ — By inversion
First subcase: $\Gamma, a : \langle G \rangle; C \vdash R \triangleright \Delta_1$ and $\Gamma; C \vdash S \triangleright \Delta_2$, $\Delta = \Delta_1, \Delta_2$ — By inversion
$\Gamma, a : \langle G \rangle; C \vdash R \triangleright \Delta_1$ and $\Gamma, a : \langle G \rangle; C \vdash S \triangleright \Delta_2$, $\Delta = \Delta_1, \Delta_2$ — By Lemma 3.4.52 (1)
$\Gamma, a : \langle G \rangle; C \vdash R \mid S \triangleright \Delta$ — By rule [Conc]

$\Gamma; C \vdash (\nu a)(R \mid S) \triangleright \Delta$ By rule [TNu]
Second subcase: $\Gamma; C \vdash R \triangleright \Delta_1, s[\hat{\mathtt{p}}_1] : T_1, ..., s[\hat{\mathtt{p}}_n] : T_\mathrm{n}$ and $\Gamma; C \vdash S \triangleright \Delta_2$,
$\Delta = \Delta_1, \Delta_2$ By inversion
$\Gamma; C \vdash R \mid S \triangleright \Delta_1, s[\hat{\mathtt{p}}_1] : T_1, ..., s[\hat{\mathtt{p}}_n] : T_\mathrm{n}, \Delta_2$ By rule [Conc]
$\Gamma; C \vdash (\nu s)(R \mid S) \triangleright \Delta$ By rule [CRes]

$\Gamma; C \vdash (\nu w)(R \mid S) \triangleright \Delta$ By assumption
First subcase: $\Gamma, a : \langle G \rangle; C \vdash R \mid S \triangleright \Delta$ By inversion
$\Gamma, a : \langle G \rangle; C \vdash R \triangleright \Delta_1$ and $\Gamma, a : \langle G \rangle; C \vdash S \triangleright \Delta_2$, $\Delta = \Delta_1, \Delta_2$ By inversion
$\Gamma; C \vdash (\nu a)R \triangleright \Delta_1$ and $\Gamma, a : \langle G \rangle; C \vdash S \triangleright \Delta_2$, $\Delta = \Delta_1, \Delta_2$ By rule [TNu]
$\Gamma; C \vdash (\nu a)R \triangleright \Delta_1$ and $\Gamma; C \vdash S \triangleright \Delta_2$, $\Delta = \Delta_1, \Delta_2$ By Lemma 3.4.53(1)
$\Gamma; C \vdash (\nu a)(R \mid S) \triangleright \Delta$ By rule [Conc]
Second subcase: $\Gamma; C \vdash R \mid S \triangleright \Delta, s[\hat{\mathtt{p}}_1] : T_1, ..., s[\hat{\mathtt{p}}_n] : T_\mathrm{n}$ By inversion
$\Gamma; C \vdash R \triangleright \Delta_1$ and $\Gamma; C \vdash S \triangleright \Delta_2$, $\Delta, s[\hat{\mathtt{p}}_1] : T_1, ..., s[\hat{\mathtt{p}}_n] : T_\mathrm{n} = \Delta_1, \Delta_2$ By inversion
First subsubcase: $s \notin dom(\Delta_1)$ $\Gamma; C \vdash R \triangleright \Delta_1, s[\mathtt{p}] : \mathsf{end}$ and
$\Gamma; C \vdash S \triangleright \Delta_2$ By Lemma 3.4.52(2)
$\Gamma; C \vdash (\nu s)R \triangleright \Delta_1$ and $\Gamma; C \vdash S \triangleright \Delta_2$ By rule [TCres]
$\Gamma; C \vdash (\nu s)R \triangleright \Delta_1$ and $\Gamma; C \vdash S \triangleright \Delta_2 \setminus s$ By Lemma 3.4.53 (2)
$\Gamma; C \vdash (\nu s)R \mid S \triangleright \Delta$ By rule [TConc]
Second subsubcase: $s \in dom(\Delta_1)$ and $s \in dom(\Delta_2)$ $\Gamma; C \vdash (\nu s)R \triangleright \Delta_1 \setminus s$
and $\Gamma; C \vdash S \triangleright \Delta_2$ By rule [TCRes]
$\Gamma; C \vdash (\nu s)R \triangleright \Delta_1 \setminus s$ and $\Gamma; C \vdash S \triangleright \Delta_2 \setminus s$ By Lemma 3.4.53 (2)
$\Gamma; C \vdash (\nu s)R \mid S \triangleright \Delta$ By rule [TConc]
Third subsubcase: $s \notin dom(\Delta_2)$ $\Gamma; C \vdash (\nu s)R \triangleright \Delta_1 \setminus s$ and
$\Gamma; C \vdash S \triangleright \Delta_2$ By rule [TCres]
$\Gamma; C \vdash (\nu s)R \mid S \triangleright \Delta$ By rule [TConc]

Case: $(\nu w)\mathbf{0} \equiv \mathbf{0}$

$\Gamma; C \vdash (\nu w)\mathbf{0} \triangleright \Delta$ By assumption
First subcase: $\Gamma, a : \langle G \rangle; C \vdash \mathbf{0} \triangleright \Delta$ By inversion
$\Gamma, a : \langle G \rangle; C \vdash \mathbf{0} \triangleright \Delta$ Δ is end only By inversion
$\Gamma, a : \langle G \rangle; C \vdash \mathbf{0} \triangleright \Delta$ By rule [TInact]
$\Gamma; C \vdash \mathbf{0} \triangleright \Delta$ By Lemma 3.4.53(1)

Second subcase: $\Gamma; C \vdash \mathbf{0} \triangleright \Delta, s[\mathsf{p}] : \mathsf{end}$, Δ is end only — By inversion
$\Gamma; C \vdash \mathbf{0} \triangleright \Delta$ — By Lemma 3.4.53(2)

$\Gamma; C \vdash \mathbf{0} \triangleright \Delta$ — By assumption
First subcase: $\Gamma, a : \langle G \rangle; C \vdash \mathbf{0} \triangleright \Delta$ — By Lemma 3.4.52(1)
$\Gamma; C \vdash (\nu a)\mathbf{0} \triangleright \Delta$ — By rule [TNu]
Second subcase: $\Gamma; C \vdash \mathbf{0} \triangleright \Delta, s[\mathsf{p}] : \mathsf{end}$ — By Lemma 3.4.52(2)
$\Gamma; C \vdash (\nu s)\mathbf{0} \triangleright \Delta$ — By rule [TCres]

Case: $\mathsf{def}\ D\ \mathsf{in}\ \mathbf{0} \equiv \mathbf{0}$

$\Gamma; C \vdash \mathsf{def}\ D\ \mathsf{in}\ \mathbf{0} \triangleright \Delta$ — By assumption
$\Gamma, X : \tilde{S}\tilde{T}; C \vdash \mathbf{0} \triangleright \Delta$ — By inversion
$\Gamma; C \vdash \mathbf{0} \triangleright \Delta$ — By Lemma 3.4.52(3)

$\Gamma; C \vdash \mathbf{0} \triangleright \Delta$ — By assumption
$\Gamma, X : \tilde{S}\tilde{T}; C \vdash \mathbf{0} \triangleright \Delta$ — By Lemma 3.4.53(3)
$\Gamma; C \vdash \mathsf{def}\ D\ \mathsf{in}\ \mathbf{0} \triangleright \Delta$ — By rule [TDef]

Case: $\mathsf{def}\ D\ \mathsf{in}\ (\nu w)R \equiv (\nu w)\mathsf{def}\ D\ \mathsf{in}\ R \quad \text{if } w \notin fn(D) \cup fc(D)$

$\Gamma; C \vdash \mathsf{def}\ D\ \mathsf{in}\ (\nu w)R \triangleright \Delta$ — By assumption
$\Gamma, X : \tilde{S}\tilde{T}; C \vdash (\nu w)R \triangleright \Delta$ and $\Gamma, X : \tilde{S}\tilde{T}, \tilde{y} : \tilde{S}; C \vdash R' \triangleright \tilde{y'} : \tilde{T}$ — By inversion
First subcase: $\Gamma, X : \tilde{S}\tilde{T}, a : \langle G \rangle; C \vdash R \triangleright \Delta$ — By inversion
$\Gamma, X : \tilde{S}\tilde{T}, \tilde{y} : \tilde{S}, a : \langle G \rangle; C \vdash R' \triangleright \tilde{y'} : \tilde{T}$ — By Lemma 3.4.52(1)
$\Gamma, a : \langle G \rangle; C \vdash \mathsf{def}\ D\ \mathsf{in}\ R \triangleright \Delta$ — By rule [TDef]
$\Gamma; C \vdash (\nu a)\mathsf{def}\ D\ \mathsf{in}\ R \triangleright \Delta$ — By rule [TNu]
Second subcase: $\Gamma, X : \tilde{S}\tilde{T}; C \vdash R \triangleright \Delta, s[\hat{\mathsf{p}}_1] : T'_1, ..., s[\hat{\mathsf{p}}_n] : T'_{\mathrm{n}}$ — By inversion
$\Gamma; C \vdash \mathsf{def}\ D\ \mathsf{in}\ R \triangleright \Delta, s[\hat{\mathsf{p}}_1] : T'_1, ..., s[\hat{\mathsf{p}}_n] : T'_{\mathrm{n}}$ — By rule [TDef]
$\Gamma; C \vdash (\nu s)\mathsf{def}\ D\ \mathsf{in}\ R \triangleright \Delta$ — By rule [TCRes]

$\Gamma; C \vdash (\nu w)\mathsf{def}\ D\ \mathsf{in}\ R \triangleright \Delta$ By assumption
First subcase: $\Gamma, a : \langle G \rangle; C \vdash \mathsf{def}\ D\ \mathsf{in}\ R \triangleright \Delta$ By inversion
$\Gamma, a : \langle G \rangle, X : \tilde{S}\tilde{T}; C \vdash R \triangleright \Delta$ and $\Gamma, X : \tilde{S}\tilde{T}, \tilde{y} : \tilde{S}, a : \langle G \rangle; C \vdash R' \triangleright \tilde{y'} : \tilde{T}$ By inversion
$\Gamma, X : \tilde{S}\tilde{T}; C \vdash (\nu a)R \triangleright \Delta$ and $\Gamma, X : \tilde{S}\tilde{T}, \tilde{y} : \tilde{S}, a : \langle G \rangle; C \vdash R' \triangleright \tilde{y'} : \tilde{T}$ By rule [TNu]
$\Gamma, X : \tilde{S}\tilde{T}; C \vdash (\nu a)R \triangleright \Delta$ and $\Gamma, X : \tilde{S}\tilde{T}, \tilde{y} : \tilde{S}; C \vdash R' \triangleright \tilde{y'} : \tilde{T}$ By Lemma 3.4.53(1)
$\Gamma; C \vdash \mathsf{def}\ D\ \mathsf{in}\ (\nu w)R \triangleright \Delta$ By rule [TDef]
Second subcase: $\Gamma; C \vdash \mathsf{def}\ D\ \mathsf{in}\ R \triangleright \Delta, s[\hat{\mathtt{p}}_1] : T'_1, ..., s[\hat{\mathtt{p}}_n] : T'_{\mathrm{n}}$ By inversion
$\Gamma, X : \tilde{S}\tilde{T}; C \vdash R \triangleright \Delta, s[\hat{\mathtt{p}}_1] : T'_1, ..., s[\hat{\mathtt{p}}_n] : T'_{\mathrm{n}}$ By inversion
$\Gamma, X : \tilde{S}\tilde{T}; C \vdash (\nu s)R \triangleright \Delta$ By rule [TCRes]
$\Gamma; C \vdash \mathsf{def}\ D\ \mathsf{in}\ (\nu s)R \triangleright \Delta$ By rule [TDef]

Case: $(\mathsf{def}\ D\ \mathsf{in}\ R) \mid S \equiv \mathsf{def}\ D\ \mathsf{in}\ (R \mid S)$ if $\mathsf{dv}(D) \cap \mathrm{fpv}(S) = \emptyset$

$\Gamma; C \vdash (\mathsf{def}\ D\ \mathsf{in}\ R) \mid S \triangleright \Delta$ By assumption
$\Gamma; C \vdash (\mathsf{def}\ D\ \mathsf{in}\ R) \triangleright \Delta_1$ and $\Gamma; C \vdash S \triangleright \Delta_2$, $\Delta = \Delta_1, \Delta_2$ By inversion
$\Gamma, X : \tilde{S}\tilde{T}; C \vdash R \triangleright \Delta_1$ and $\Gamma; C \vdash S \triangleright \Delta_2$, $\Delta = \Delta_1, \Delta_2$ By inversion
$\Gamma, X : \tilde{S}\tilde{T}; C \vdash R \triangleright \Delta_1$ and $\Gamma, X : \tilde{S}\tilde{T}; C \vdash S \triangleright \Delta_2$, $\Delta = \Delta_1, \Delta_2$ By Lemma 3.4.52(3)
$\Gamma, X : \tilde{S}\tilde{T}; C \vdash R \mid S \triangleright \Delta$ By rule [TConc]
$\Gamma; C \vdash \mathsf{def}\ D\ \mathsf{in}\ (R \mid S) \triangleright \Delta$ By rule [TDef]

$\Gamma; C \vdash \mathsf{def}\ D\ \mathsf{in}\ (R \mid S) \triangleright \Delta$ By assumption
$\Gamma, X : \tilde{S}\tilde{T}; C \vdash R \mid S \triangleright \Delta$ By inversion
$\Gamma, X : \tilde{S}\tilde{T}; C \vdash R \triangleright \Delta_1$ and $\Gamma, X : \tilde{S}\tilde{T}; C \vdash S \triangleright \Delta_2$, $\Delta = \Delta_1, \Delta_2$ By inversion
$\Gamma; C \vdash (\mathsf{def}\ D\ \mathsf{in}\ R) \triangleright \Delta_1$ and $\Gamma, X : \tilde{S}\tilde{T}; C \vdash S \triangleright \Delta_2$, $\Delta = \Delta_1, \Delta_2$ By rule [TDef]
$\Gamma; C \vdash (\mathsf{def}\ D\ \mathsf{in}\ R) \triangleright \Delta_1$ and $\Gamma; C \vdash S \triangleright \Delta_2$, $\Delta = \Delta_1, \Delta_2$ By Lemma 3.4.53(3)
$\Gamma; C \vdash (\mathsf{def}\ D\ \mathsf{in}\ R) \mid S \triangleright \Delta$ By rule [TConc]

Case: $s : (\hat{\mathtt{q}}, \hat{\mathtt{p}}, z) \cdot (\hat{\mathtt{q}}', \hat{\mathtt{p}}', z') \cdot h \equiv s : (\hat{\mathtt{q}}', \hat{\mathtt{p}}', z') \cdot (\hat{\mathtt{q}}, \hat{\mathtt{p}}, z) \cdot h$ if $\hat{\mathtt{p}} \neq \hat{\mathtt{p}}'$ or $\hat{\mathtt{q}} \neq \hat{\mathtt{q}}'$

$\Gamma; C \vdash s : (\hat{\mathtt{q}}, \hat{\mathtt{p}}, z) \cdot (\hat{\mathtt{q}}', \hat{\mathtt{p}}', z') \cdot h \triangleright \Delta$ By assumption
$\Delta = s[\hat{\mathtt{q}}] :!\langle \hat{\mathtt{p}}, S \rangle; s[\hat{\mathtt{q}}'] :!\langle \hat{\mathtt{p}}', S' \rangle; \Delta'$ By rule [TRSend]
$\Gamma; C \vdash s : (\hat{\mathtt{q}}', \hat{\mathtt{p}}', z') \cdot (\hat{\mathtt{q}}, \hat{\mathtt{p}}, z) \cdot h \triangleright (s[\hat{\mathtt{q}}'] :!\langle \hat{\mathtt{p}}', S' \rangle, s[\hat{\mathtt{q}}] :!\langle \hat{\mathtt{p}}, S \rangle); \Delta'$ By rule [TRSend]
Note that $s[\hat{\mathtt{q}}] :!\langle \hat{\mathtt{p}}, S \rangle; s[\hat{\mathtt{q}}'] :!\langle \hat{\mathtt{p}}', S' \rangle; \Delta' = (s[\hat{\mathtt{q}}] :!\langle \hat{\mathtt{p}}, S \rangle, s[\hat{\mathtt{q}}'] :!\langle \hat{\mathtt{p}}', S' \rangle); \Delta' \rightarrow$

$(s[\hat{\mathtt{q}}] :!\langle \hat{\mathtt{p}}, S\rangle, s[\hat{\mathtt{q}}'] :!\langle \hat{\mathtt{p}}', S'\rangle); \Delta' = (s[\hat{\mathtt{q}}'] :!\langle \hat{\mathtt{p}}', S'\rangle, s[\hat{\mathtt{q}}] :!\langle \hat{\mathtt{p}}, S\rangle); \Delta'$

The other case is symmetric to the above one. □

Lemma 3.4.55. 1. If $C, n : \mathtt{T} \vdash \mathtt{t} \geq \mathrm{n}'$ and $C \vdash \mathrm{n} \geq \min(\mathtt{T})$ then $C\{\mathrm{n}/n\} \vdash \mathtt{t}\{\mathrm{n}/n\} \geq \mathrm{n}'$.

2. If $C, n : \mathtt{T} \vdash \mathtt{t}$ and $C \vdash \mathrm{n} \geq \min(\mathtt{T})$ then $C\{\mathrm{n}/n\} \vdash \mathtt{t}\{\mathrm{n}/n\}$.

Proof. (1) is by induction on the typing judgment $C \vdash \mathtt{t} \geq \mathrm{n}$. We present the most appealing cases, including those most difficult.

$C, n : \mathtt{T} \vdash \mathtt{t} + \mathrm{n}'' \geq \mathrm{n}'$ and $C \vdash \mathrm{n} \geq \min(\mathtt{T})$	By assumption
$C, n : \mathtt{T} \vdash \mathtt{t} \geq \mathrm{n}' - \mathrm{n}''$ and $C \vdash \mathrm{n} \geq \min(\mathtt{T})$	By inversion
$C\{\mathrm{n}/n\} \vdash \mathtt{t}\{\mathrm{n}/n\} \geq \mathrm{n}' - \mathrm{n}''$	By i.h.
$C\{\mathrm{n}/n\} \vdash \mathtt{t}\{\mathrm{n}/n\} + \mathrm{n}'' \geq \mathrm{n}'$	By rule

The other cases $\mathtt{t} - \mathrm{n}'' \geq \mathrm{n}'$, $\mathrm{n}''^{\mathtt{t}} \geq \mathrm{n}'$, $\mathtt{t} * \mathrm{n}'' \geq \mathrm{n}'$ are symmetric.

$C, n : \mathtt{T} \vdash \mathrm{n}' \geq \mathrm{n}''$ and $C \vdash \mathrm{n} \geq \min(\mathtt{T})$	By assumption
$\mathrm{n}' \geq \mathrm{n}''$	By inversion
$C\{\mathrm{n}/n\} \vdash \mathrm{n}' \geq \mathrm{n}''$	By rule

$C, n : \mathtt{T} \vdash n' \geq \mathrm{n}'$ and $C \vdash \mathrm{n} \geq \min(\mathtt{T})$	By assumption
$C, n : \mathtt{T} \vdash n' : \{n' \mid n' \geq \mathrm{n}''\}$, $\mathrm{n}'' \geq \mathrm{n}'$ and $C \vdash \mathrm{n} \geq \min(\mathtt{T})$	By inversion
$C\{\mathrm{n}/n\} \vdash n' : \{n' \mid n' \geq \mathrm{n}''\}$, $\mathrm{n}'' \geq \mathrm{n}'$	By i.h.
$C\{\mathrm{n}/n\} \vdash n' \geq \mathrm{n}'$	By rule

(2) is by induction on the typing judgment $C \vdash \mathtt{t}$. We present the most appealing cases, including those most difficult.

$C, n : \mathtt{T} \vdash \mathrm{n}'$ and $C \vdash \mathrm{n} \geq \min(\mathtt{T})$	By assumption
$C\{\mathrm{n}/n\} \vdash \mathrm{n}$‘	By rule

$C, n : \mathtt{T} \vdash n'$ and $C \vdash \mathrm{n} \geq \min(\mathtt{T})$ By assumption
$n' \in dom(C)$ By inversion
$C\{\mathrm{n}/n\} \vdash n'$ By rule

$C, n : \mathtt{T} \vdash \mathtt{t} * \mathrm{n}'$ and $C \vdash \mathrm{n} \geq \min(\mathtt{T})$ By assumption
$C, n : \mathtt{T} \vdash \mathtt{t}$ and$C \vdash \mathrm{n} \geq \min(\mathtt{T})$ By inversion
$C\{\mathrm{n}/n\} \vdash \mathtt{t}\{\mathrm{n}/n\}$ By i.h.
$C\{\mathrm{n}/n\} \vdash \mathtt{t}\{\mathrm{n}/n\} * \mathrm{n}'$ By rule

Other cases are symmetric. □

Lemma 3.4.56 (Type Preservation Under Substitution)**.** 1. If $\Gamma; C, n : \mathtt{T} \vdash E \rhd \tau$ and $C \vdash \mathrm{n} \geq \min(\mathtt{T})$, then $\Gamma\{\mathrm{n}/n\}; C\{\mathrm{n}/n\} \vdash E\{\mathrm{n}/n\} \vdash \tau\{\mathrm{n}/n\}$.

2. If $\Gamma; C, i : \mathtt{I} \vdash E \rhd \tau$ and $\mathrm{n} \leq \max(\mathtt{I})$, then $\Gamma\{\mathrm{n}/i\}; C \vdash E\{\mathrm{n}/i\} \rhd \tau\{\mathrm{n}/i\}$.

3. If $\Gamma, C \vdash R \rhd \Delta, y : T$ then $\Gamma; C \vdash R\{s[\hat{\mathtt{p}}]/y\} \rhd \Delta, s[\hat{\mathtt{p}}] : T$.

4. If $\Gamma, \tilde{x} : \tilde{S}; C \vdash R \rhd \Delta$ and $\Gamma; C \vdash \tilde{v} : \tilde{S}$ then $\Gamma; C \vdash R[\tilde{v}/\tilde{x}] \rhd \Delta$.

Proof. (1) is by induction on the typing judgement $\Gamma; C, n : \mathtt{T} \vdash E \rhd \tau$. We present the most appealing cases, including those most difficult.

$\Gamma, u : \langle G \rangle; C, n : \mathtt{T} \vdash u : \langle G \rangle$ and $C \vdash \mathrm{n} \geq \min(\mathtt{T})$ By assumption
$\Gamma\{\mathrm{n}/n\}, u : \langle G \rangle\{\mathrm{n}/n\}; C\{\mathrm{n}/n\} \vdash u : \langle G \rangle\{\mathrm{n}/n\}$ By rule [TName]
Note that $\langle G \rangle\{\mathrm{n}/n\} = \langle G\{\mathrm{n}/n\} \rangle$

$\Gamma; C, n : \mathtt{T} \vdash \mathbf{true}, \mathbf{false} : \mathsf{bool}$ and $C \vdash \mathrm{n} \geq \min(\mathtt{T})$ By assumption
$\Gamma\{\mathrm{n}/n\}; C\{\mathrm{n}/n\} \vdash \mathbf{true}, \mathbf{false} : \mathsf{bool}$ By rule [TBool]
where $\mathsf{bool}\{\mathrm{n}/n\} = \mathsf{bool}$.

$\Gamma; C, n : \mathtt{T} \vdash e_1 \mathtt{or}\ e_2 : \mathsf{bool}$ and $C \vdash \mathrm{n} \geq \min(\mathtt{T})$ By assumption
$\Gamma; C, n : \mathtt{T} \vdash e_i \rhd \mathsf{bool}$ and $C \vdash \mathrm{n} \geq \min(\mathtt{T})$ By inversion
$\Gamma\{\mathrm{n}/n\}; C\{\mathrm{n}/n\} \vdash e_i\{\mathrm{n}/n\} \rhd \mathsf{bool}$ By i.h.
$\Gamma\{\mathrm{n}/n\}; C\{\mathrm{n}/n\} \vdash e_1\{\mathrm{n}/n\} \mathtt{or}\ e_2\{\mathrm{n}/n\} : \mathsf{bool}$ By rule [TOr]

$\Gamma; C, n : \mathtt{T} \vdash \lambda m.E \rhd \Pi m : \mathtt{T}'.\tau$ and $C \vdash \mathrm{n} \geq \min(\mathtt{T})$ By assumption
$\Gamma; C, n : \mathtt{T}, m : \mathtt{T}' \vdash E \rhd \tau$ and $C \vdash \mathrm{n} \geq \min(\mathtt{T})$ By inversion
$\Gamma\{\mathrm{n}/n\}; C\{\mathrm{n}/n\}, m : \mathtt{T}'\{\mathrm{n}/n\} \vdash E\{\mathrm{n}/n\} \vdash \tau\{\mathrm{n}/n\}$ By i.h.
$\Gamma\{\mathrm{n}/n\}; C\{\mathrm{n}/n\}, \vdash \lambda m.E\{\mathrm{n}/n\} \vdash \Pi m : \mathtt{T}'\{\mathrm{n}/n\}.\tau\{\mathrm{n}/n\}$ By rule [TFun]
where $(\Pi m : \mathtt{T}.\tau)\{\mathrm{n}/n\} = \Pi m : \mathtt{T}\{\mathrm{n}/n\}.(\tau\{\mathrm{n}/n\})$ and $\mathtt{T}\{\mathrm{n}/n\} = \mathtt{T}$.

$\Gamma; C, n : \mathtt{T} \vdash E\ \mathtt{t} \rhd \tau$ and $C \vdash \mathrm{n} \geq \min(\mathtt{T})$ By assumption
$\Gamma; C, n : \mathtt{T} \vdash E \rhd \Pi m : \mathtt{T}' : \tau$, $C, n : \mathtt{T} \vdash \mathtt{t} \geq \min(\mathtt{T}')$, $C \vdash \mathrm{n} \geq \min(\mathtt{T})$ By inversion
$\Gamma\{\mathrm{n}/n\}; C\{\mathrm{n}/n\} \vdash E\{\mathrm{n}/n\} \vdash \Pi m : \mathtt{T}'.\tau\{\mathrm{n}/n\}$
$C\{\mathrm{n}/n\} \vdash \mathtt{t}\{\mathrm{n}/n\} \geq \min(\mathtt{T}')\{\mathrm{n}/n\}$ By i.h.and Lemma 3.4.55(1)
$\Gamma\{\mathrm{n}/n\}; C\{\mathrm{n}/n\}, \vdash E\{\mathrm{n}/n\}\ \mathtt{t}\{n/\mathrm{n}\} \vdash \tau\{\mathrm{n}/n\}$ By rule [TAppF]
where $(\Pi m : \mathtt{T}.\tau)\{\mathrm{n}/n\} = \Pi m : \mathtt{T}.(\tau\{\mathrm{n}/n\})$.

$\Gamma; C, n : \mathtt{T} \vdash \mathbf{R}\ S\lambda i.\lambda X.R \rhd \Pi i : \mathtt{I}.\Delta$ and $C \vdash \mathrm{n} \geq \min(\mathtt{T})$ By assumption
$\Gamma; C, n : \mathtt{T} \vdash S \rhd \Delta\ 0$, $\Gamma, X : \Delta\ i; C, n : \mathtt{T}, i : \mathtt{I} \vdash R \rhd \Delta\ i + 1$,
$C \vdash \mathrm{n} \geq \min(\mathtt{T})$ By inversion
$\Gamma\{\mathrm{n}/n\}; C\{\mathrm{n}/n\} \vdash S\{\mathrm{n}/n\} \rhd \Delta\{\mathrm{n}/n\}\ 0$, $\Gamma\{\mathrm{n}/n\}, X : \Delta\{\mathrm{n}/n\}\ i; C\{\mathrm{n}/n\} i : \mathtt{I}\{\mathrm{n}/n\}$
$\vdash R\{\mathrm{n}/n\} \rhd \Delta\{\mathrm{n}/n\}\ i + 1$ By i.h.
$\Gamma\{\mathrm{n}/n\}; C\{\mathrm{n}/n\} \vdash \mathbf{R}\ S\{\mathrm{n}/n\}\lambda i.\lambda X.R\{\mathrm{n}/n\} \rhd \Pi i : \mathtt{I}\{\mathrm{n}/n\}.(\Delta\{\mathrm{n}/n\})$ By rule [TPRec]
Note that $(\Pi i : \mathtt{I}.\Delta)\{\mathrm{n}/n\} = \Pi i : \mathtt{I}\{\mathrm{n}/n\}.\Delta\{\mathrm{n}/n\}$

$\Gamma; C, n : \mathtt{T} \vdash R\ \mathtt{t} \rhd \Delta\ \mathtt{t}$ and $C \vdash \mathrm{n} \geq \min(\mathtt{T})$ By assumption
$C, n : \mathtt{T} \vdash \mathtt{t} \quad \Gamma; C, n : \mathtt{T} \vdash R \rhd \Pi i{:}\{i \mid i{\in}\mathsf{nat}, 0{\leq}i{\leq}\mathtt{t}{-}1\}.\Delta$
and $C \vdash \mathrm{n} \geq \min(\mathtt{T})$ By inversion
$C\{\mathrm{n}/n\} \vdash \mathtt{t}\{\mathrm{n}/n\} \quad \Gamma\{\mathrm{n}/n\}; C\{\mathrm{n}/n\} \vdash R\{\mathrm{n}/n\} \rhd$
$\Pi i{:}\{i \mid i{\in}\mathsf{nat}, 0{\leq}i{\leq}\mathtt{t}\{\mathrm{n}/n\}{-}1\}.\Delta\{\mathrm{n}/n\}$ By Lemma 3.4.55(2) and i.h.
Note that $\{i \mid i{\in}\mathsf{nat}, 0{\leq}i{\leq}\mathtt{t}{-}1\}\{\mathrm{n}/n\} = \{i \mid i{\in}\mathsf{nat}, 0{\leq}i{\leq}\mathtt{t}\{\mathrm{n}/n\}{-}1\}$
$\Gamma\{\mathrm{n}/n\}; C\{\mathrm{n}/n\} \vdash R\{\mathrm{n}/n\}\ \mathtt{t} \rhd \Delta\{\mathrm{n}/n\}\ \mathtt{t}\{\mathrm{n}/n\}$ By rule [TAppR]

$\Gamma; C, n : \mathtt{T} \vdash \bar{u}[\mathsf{p}_0, \mathsf{p}_1, p](y).R \rhd \Delta$ and $C \vdash \mathrm{n} \geq \min(\mathtt{T})$ By assumption
$\Gamma; C, n : \mathtt{T} \vdash u : \langle G \rangle \quad \emptyset; C, n : \mathtt{T} \vdash G \blacktriangleright \mathsf{Type}, C, n : \mathtt{T} \vdash \mathsf{p}_0, \mathsf{p}_1, p$
$C, n : \mathtt{T} \vdash \mathsf{pid}(G) = \{\mathsf{p}_0, \mathsf{p}_1, p\}, \quad \Gamma; C, n : \mathtt{T} \vdash R \rhd \Delta, y : \xi(G \upharpoonright \mathsf{p}_0)$ By inversion
$\Gamma\{\mathrm{n}/n\}; C \vdash u : \langle G\{\mathrm{n}/n\} \rangle \quad \emptyset; C\{\mathrm{n}/n\} \vdash G\{\mathrm{n}/n\} \blacktriangleright \mathsf{Type}, C\{\mathrm{n}/n\} \vdash (\mathsf{p}_0, \mathsf{p}_1, p)\{\mathrm{n}/n\}$
Note that $(\mathsf{p}_0, \mathsf{p}_1, p)\{\mathrm{n}/n\} = \mathsf{p}_0\{\mathrm{n}/n\}, \mathsf{p}_1\{\mathrm{n}/n\}, p\{\mathrm{n}/n\}$
$C\{\mathrm{n}/n\} \vdash \mathsf{pid}(G\{\mathrm{n}/n\}) = \{\mathsf{p}_0\{\mathrm{n}/n\}, \mathsf{p}_1\{\mathrm{n}/n\}, p\{\mathrm{n}/n\}\}$

$\Gamma\{\mathrm{n}/n\}; C\{\mathrm{n}/n\} \vdash R\{\mathrm{n}/n\} \triangleright \Delta\{\mathrm{n}/n\}, y : \xi(G\{\mathrm{n}/n\} \upharpoonright \mathtt{p}_0\{\mathrm{n}/n\})$ By i.h.
$\Gamma\{\mathrm{n}/n\}; C\{\mathrm{n}/n\} \vdash \bar{u}[\mathtt{p}_0\{\mathrm{n}/n\}, \mathtt{p}_1\{\mathrm{n}/n\}, p\{\mathrm{n}/n\}](y).R\{\mathrm{n}/n\} \triangleright \Delta\{\mathrm{n}/n\}$ By rule [TMCast]
Note that $(\bar{u}[\mathtt{p}_0, \mathtt{p}_1, p](y).R)\{\mathrm{n}/n\} = \bar{u}[\mathtt{p}_0\{\mathrm{n}/n\}, \mathtt{p}_1\{\mathrm{n}/n\}, p\{\mathrm{n}/n\}](y).R\{\mathrm{n}/n\}$

The other case is symmetric.

$\Gamma; C, n : \mathtt{T} \vdash k!\langle \mathtt{p}, e\rangle; R \triangleright \Delta, k :!\langle \mathtt{p}, S\rangle; T$ and $C \vdash \mathrm{n} \geq \min(\mathtt{T})$ By assumption
$\Gamma; C, n : \mathtt{T} \vdash e \triangleright S \quad C, n : \mathtt{T} \vdash \mathtt{p} \quad \Gamma; C, n : \mathtt{T} \vdash R \triangleright \Delta, k : T$
and $C \vdash \mathrm{n} \geq \min(\mathtt{T})$ By inversion
$\Gamma\{\mathrm{n}/n\}; C\{\mathrm{n}/n\} \vdash e\{\mathrm{n}/n\} \triangleright S\{\mathrm{n}/n\} \quad C\{\mathrm{n}/n\} \vdash \mathtt{p}\{\mathrm{n}/n\}$
$\Gamma; C\{\mathrm{n}/n\} \vdash R\{\mathrm{n}/n\} \triangleright \Delta\{\mathrm{n}/n\}, k : T\{\mathrm{n}/n\}$ By Lemma 3.4.55(2) and i.h.
$\Gamma\{\mathrm{n}/n\}; C\{\mathrm{n}/n\} \vdash k!\langle \mathtt{p}\{\mathrm{n}/n\}, e\{\mathrm{n}/n\}\rangle; R\{\mathrm{n}/n\} \triangleright$
$\Delta\{\mathrm{n}/n\}, k :!\langle \mathtt{p}\{\mathrm{n}/n\}, S\{\mathrm{n}/n\}\rangle; T\{\mathrm{n}/n\}$ By rule [TSend]
Note that $\Delta, k :!\langle \mathtt{p}, S\rangle; T)\{\mathrm{n}/n\} =!\langle \mathtt{p}\{\mathrm{n}/n\}, S\{\mathrm{n}/n\}\rangle; T\{\mathrm{n}/n\}$

Cases: [TRcv], [TSel], [TBr] are symmetric.

$\Gamma; C, n : \mathtt{T} \vdash (\nu a)R \triangleright \Delta$ and $C \vdash \mathrm{n} \geq \min(\mathtt{T})$ By assumption
$\Gamma, a : \langle G\rangle; C, n : \mathtt{T} \vdash R \triangleright \Delta$ and $C \vdash \mathrm{n} \geq \min(\mathtt{T})$ By inversion
$\Gamma\{\mathrm{n}/n\}, a : \langle G\{\mathrm{n}/n\}\rangle; C\{\mathrm{n}/n\} \vdash R\{\mathrm{n}/n\} \triangleright \Delta\{\mathrm{n}/n\}$ By i.h.
$\Gamma\{\mathrm{n}/n\}; C\{\mathrm{n}/n\} \vdash (\nu a)R\{\mathrm{n}/n\} \triangleright \Delta\{\mathrm{n}/n\}$ By rule [Tnu]
where $(\nu a)R\{\mathrm{n}/n\} = (\nu a)R\{\mathrm{n}/n\}$.

$\Gamma; C, n : \mathtt{T} \vdash R \mid S \triangleright \Delta, \Delta'$ and $C \vdash \mathrm{n} \geq \min(\mathtt{T})$ By assumption
$\Gamma; C, n : \mathtt{T} \vdash R \triangleright \Delta \quad \Gamma; C, n : \mathtt{T} \vdash S \triangleright \Delta'$ and $C \vdash \mathrm{n} \geq \min(\mathtt{T})$ By inversion
$\Gamma\{\mathrm{n}/n\}; C\{\mathrm{n}/n\} \vdash R\{\mathrm{n}/n\} \triangleright \Delta\{\mathrm{n}/n\} \quad \Gamma; C\{\mathrm{n}/n\} \vdash S\{\mathrm{n}/n\} \triangleright \Delta'\{\mathrm{n}/n\}$ By i.h.
$\Gamma\{\mathrm{n}/n\}; C\{\mathrm{n}/n\} \vdash R\{\mathrm{n}/n\} \mid S\{\mathrm{n}/n\} \triangleright \Delta\{\mathrm{n}/n\}, \Delta'\{\mathrm{n}/n\}$ By rule [TConc]
where $R\{\mathrm{n}/n\} \mid S\{\mathrm{n}/n\} = (R \mid S)\{\mathrm{n}/n\}$ and $\Delta\{\mathrm{n}/n\}, \Delta'\{\mathrm{n}/n\} = (\Delta, \Delta')\{\mathrm{n}/n\}$.

Cases [TVar], [TInact], [TIF] are trivial.

$\Gamma; C, n : \mathtt{T} \vdash R \triangleright \Delta'$ and $C \vdash \mathrm{n} \geq \min(\mathtt{T})$ By assumption
$\Gamma; C, n : \mathtt{T} \vdash R \triangleright \Delta \quad \Delta <: \Delta'$ and $C \vdash \mathrm{n} \geq \min(\mathtt{T})$ By inversion
$\Gamma\{\mathrm{n}/n\}; C\{\mathrm{n}/n\} \vdash R\{\mathrm{n}/n\} \triangleright \Delta\{\mathrm{n}/n\} \quad \Delta <: \Delta'$ By i.h.

$\Delta <: \Delta' \Rightarrow \Delta\{\mathrm{n}/n\} <: \Delta'\{\mathrm{n}/n\}$ By Definition 3.4.49
$\Gamma\{\mathrm{n}/n\}; C\{\mathrm{n}/n\} \vdash R\{\mathrm{n}/n\} \triangleright \Delta'\{\mathrm{n}/n\}$ By rule [<:]

$\Gamma; C, n : \mathtt{T} \vdash (\nu s)R \triangleright \Delta$ and $C \vdash \mathrm{n} \geq \min(\mathtt{T})$ By assumption
$\Gamma; C, n : \mathtt{T} \vdash R \triangleright \Delta, s[\hat{\mathtt{p}}_1] : T_1, ..., s[\hat{\mathtt{p}}_n] : T_n$ and $C \vdash \mathrm{n} \geq \min(\mathtt{T})$ By inversion
$\Gamma\{\mathrm{n}/n\}; C\{\mathrm{n}/n\} \vdash R\{\mathrm{n}/n\} \triangleright \Delta\{\mathrm{n}/n\}, s[\hat{\mathtt{p}}_1] : T_1\{\mathrm{n}/n\}, ..., s[\hat{\mathtt{p}}_n] : T_{\mathrm{n}}\{\mathrm{n}/n\}$ By i.h.
$\Gamma\{\mathrm{n}/n\}; C\{\mathrm{n}/n\} \vdash (\nu s)R\{\mathrm{n}/n\} \triangleright \Delta\{\mathrm{n}/n\}$ By rule [TCres]
where $(\nu s)R\{\mathrm{n}/n\} = ((\nu s)R)\{\mathrm{n}/n\}$.

$\Gamma, X : \tilde{S}\tilde{T}; C, n : \mathtt{T} \vdash X\langle \tilde{e}, \tilde{k} \rangle \triangleright \Delta, \tilde{k} : \tilde{T}$ and $C \vdash \mathrm{n} \geq \min(\mathtt{T})$ By assumption
$\Gamma; C, n : \mathtt{T} \vdash \tilde{e} : \tilde{S} \quad \Delta$ end only and $C \vdash \mathrm{n} \geq \min(\mathtt{T})$ By inversion
$\Gamma\{\mathrm{n}/n\}; C\{\mathrm{n}/n\} \vdash \tilde{e}\{\mathrm{n}/n\} : \tilde{S}\{\mathrm{n}/n\} \quad \Delta$ end only By i.h.
$\Gamma\{\mathrm{n}/n\}, X : \tilde{S}\{\mathrm{n}/n\}\tilde{T}; C\{\mathrm{n}/n\} \vdash X\langle \tilde{e}\{\mathrm{n}/n\}, \tilde{k} \rangle \triangleright \Delta\{\mathrm{n}/n\}, \tilde{k} : \tilde{T}$ By rule [TVar]

$\Gamma; C, n : \mathtt{T} \vdash \mathsf{def}\ X(\tilde{y}, \tilde{y'}) = R\ \mathsf{in}\ R' \triangleright \Delta$ and $C \vdash \mathrm{n} \geq \min(\mathtt{T})$ By assumption
$\Gamma, X : \tilde{S}\tilde{T}, \tilde{y} : \tilde{S}; C, n : \mathtt{T} \vdash R \triangleright \tilde{y'} : \tilde{T} \quad \Gamma, X : \tilde{S}\tilde{T}; C, n : \mathtt{T} \vdash R' \triangleright \Delta$
and $C \vdash \mathrm{n} \geq \min(\mathtt{T})$ By inversion
$\Gamma\{\mathrm{n}/n\}, X : \tilde{S}\{\mathrm{n}/n\}\tilde{T}\{\mathrm{n}/n\}, \tilde{y} : \tilde{S}\{\mathrm{n}/n\}; C\{\mathrm{n}/n\} \vdash R\{\mathrm{n}/n\} \triangleright \tilde{y'} : \tilde{T}\{\mathrm{n}/n\}$
$\Gamma\{\mathrm{n}/n\}, X : \tilde{S}\{\mathrm{n}/n\}\tilde{T}\{\mathrm{n}/n\}; C\{\mathrm{n}/n\} \vdash R'\{\mathrm{n}/n\} \triangleright \Delta\{\mathrm{n}/n\}$ By i.h.
$\Gamma\{\mathrm{n}/n\}; C\{\mathrm{n}/n\} \vdash \mathsf{def}\ X(\tilde{y}, \tilde{y'}) = R\{\mathrm{n}/n\}\ \mathsf{in}\ R'\{\mathrm{n}/n\} \triangleright \Delta\{\mathrm{n}/n\}$ By rule [TDef]
where $\mathsf{def}\ X(\tilde{y}, \tilde{y'}) = R\{\mathrm{n}/n\}\ \mathsf{in}\ R'\{\mathrm{n}/n\} = (\mathsf{def}\ X(\tilde{y}, \tilde{y'}) = R\ \mathsf{in}\ R')\{\mathrm{n}/n\}$

(2) is by induction on the typing judgement $\Gamma; C, i : \mathtt{I} \vdash E \triangleright \tau$. We present the most appealing cases, including those most difficult.

$\Gamma, u : \langle G \rangle; C, i : \mathtt{I} \vdash u : \langle G \rangle$ and $\mathrm{n} \leq \max(\mathtt{I})$ By assumption
$\Gamma\{\mathrm{n}/i\}, u : \langle G \rangle\{\mathrm{n}/i\}; C \vdash u : \langle G \rangle\{\mathrm{n}/i\}$ By rule [TName]
Note that $\langle G \rangle\{\mathrm{n}/i\} = \langle G \rangle$

$\Gamma; C, i : \mathtt{I} \vdash \mathbf{true}, \mathbf{false} : \mathsf{bool}$ and $\mathrm{n} \leq \max(\mathtt{I})$ By assumption
$\Gamma\{\mathrm{n}/i\}; C \vdash \mathbf{true}, \mathbf{false} : \mathsf{bool}\{\mathrm{n}/i\}$ By rule [TBool]
where $\mathsf{bool}\{\mathrm{n}/i\} = \mathsf{bool}$.

$\Gamma; C, i : \mathtt{I} \vdash e_1 \mathtt{or}\ e_2 \colon \mathsf{bool}$ and $\mathrm{n} \leq \max(\mathtt{I})$ By assumption
$\Gamma; C, i : \mathtt{I} \vdash e_i \triangleright \mathsf{bool}$ and $\mathrm{n} \leq \max(\mathtt{I})$ By inversion
$\Gamma\{\mathrm{n}/i\}; C \vdash e_i\{\mathrm{n}/i\} \triangleright \mathsf{bool}\{\mathrm{n}/i\}$ By i.h.
$\Gamma\{\mathrm{n}/i\}; C \vdash e_1\{\mathrm{n}/i\} \mathtt{or}\ e_2\{\mathrm{n}/i\} \colon \mathsf{bool}\{\mathrm{n}/i\}$ By rule [TOr]

$\Gamma; C, i : \mathtt{I} \vdash \lambda m.E \triangleright \Pi m : \mathtt{T}'.\tau$ and $\mathrm{n} \leq \max(\mathtt{I})$ By assumption
$\Gamma; C, i : \mathtt{I}, m : \mathtt{T}' \vdash E \triangleright \tau$ and $\mathrm{n} \leq \max(\mathtt{I})$ By inversion
$\Gamma\{\mathrm{n}/i\}; C, m : \mathtt{T}' \vdash E\{\mathrm{n}/i\} \vdash \tau\{\mathrm{n}/i\}$ By i.h.
$\Gamma\{\mathrm{n}/i\}; C \vdash \lambda m.E\{\mathrm{n}/i\} \vdash \Pi m : \mathtt{T}'.\tau\{\mathrm{n}/i\}$ By rule [TFun]
where $(\Pi m : \mathtt{T}.\tau)\{\mathrm{n}/i\} = \Pi m : \mathtt{T}.(\tau\{\mathrm{n}/i\})$.

$\Gamma; C, i : \mathtt{I} \vdash E\ \mathtt{t} \triangleright \tau$ and $\mathrm{n} \leq \max(\mathtt{I})$ By assumption
$\Gamma; C, i : \mathtt{I} \vdash E \triangleright \Pi m : \mathtt{T}' : \tau$, $C, i : \mathtt{I} \vdash \mathtt{t} \geq \min(\mathtt{T}')$, $\mathrm{n} \leq \max(\mathtt{I})$ By inversion
$\Gamma\{\mathrm{n}/i\}; C \vdash E\{\mathrm{n}/i\} \vdash \Pi m : \mathtt{T}'.\tau\{\mathrm{n}/i\}$, $C \vdash \mathtt{t} \geq \min(\mathtt{T}')$ By i.h.
$\Gamma\{\mathrm{n}/i\}; C, \vdash E\{\mathrm{n}/i\}\ \mathtt{t} \vdash \tau\{\mathrm{n}/i\}$ By rule [TAppF]
where $(\Pi m : \mathtt{T}.\tau)\{\mathrm{n}/i\} = \Pi m : \mathtt{T}.(\tau\{\mathrm{n}/i\})$.

$\Gamma; C, i : \mathtt{I} \vdash \mathbf{R}\ S\lambda j.\lambda X.R \triangleright \Pi j : \mathtt{I}'.\Delta$ and $\mathrm{n} \leq \max(\mathtt{I})$ By assumption
$\Gamma; C, i : \mathtt{I} \vdash S \triangleright \Delta\ 0$, $\Gamma, X : \Delta\ j; C, i : \mathtt{I}, j : \mathtt{I}' \vdash R \triangleright \Delta\ j + 1$,
$\mathrm{n} \leq \max(\mathtt{I})$ By inversion
$\Gamma\{\mathrm{n}/i\}; C \vdash S\{\mathrm{n}/i\} \triangleright \Delta\{\mathrm{n}/i\}\ 0$, $\Gamma\{\mathrm{n}/i\}, X : \Delta\{\mathrm{n}/i\}\ j; C, j : \mathtt{I}' \vdash R\{\mathrm{n}/i\} \triangleright$
$\Delta\{\mathrm{n}/i\}\ j + 1$ By i.h.
$\Gamma\{\mathrm{n}/i\}; C \vdash \mathbf{R}\ S\{\mathrm{n}/i\}\lambda j.\lambda X.R\{\mathrm{n}/i\} \triangleright \Pi j : \mathtt{I}'.(\Delta\{\mathrm{n}/i\})$ By rule [TPRec]

$\Gamma; C, i : \mathtt{I} \vdash R\ \mathtt{t} \triangleright \Delta\ \mathtt{t}$ and $\mathrm{n} \leq \max(\mathtt{I})$ By assumption
$C, i : \mathtt{I} \vdash \mathtt{t} \quad \Gamma; C, i : \mathtt{I} \vdash R \triangleright \Pi j{:}\{j \mid j{\in}\mathsf{nat}, 0{\leq}j{\leq}\mathtt{t}{-}1\}.\Delta$ and $\mathrm{n} \leq \max(\mathtt{I})$ By inversion
$C \vdash \mathtt{t} \quad \Gamma\{\mathrm{n}/i\}; C \vdash R\{\mathrm{n}/i\} \triangleright \Pi j{:}\{j \mid j{\in}\mathsf{nat}, 0{\leq}j{\leq}\mathtt{t}{-}1\}.\Delta\{\mathrm{n}/i\}$ By i.h.
$\Gamma\{\mathrm{n}/i\}; C \vdash R\{\mathrm{n}/i\}\ \mathtt{t} \triangleright \Delta\{\mathrm{n}/i\}\ \mathtt{t}$ By rule [TAppR]

$\Gamma; C, i : \mathtt{I} \vdash \bar{u}[\mathtt{p}_0, \mathtt{p}_1, p](y).R \triangleright \Delta$ and $\mathrm{n} \leq \max(\mathtt{I})$ By assumption
$\Gamma \vdash u : \langle G \rangle \quad \emptyset; C, i : \mathtt{I} \vdash G \blacktriangleright \mathsf{Type}, C, i : \mathtt{I} \vdash \mathtt{p}_0, \mathtt{p}_1, p \quad C, i : \mathtt{I} \vdash \mathsf{pid}(G) = \{\mathtt{p}_0, \mathtt{p}_1, p\}$,
$\Gamma; C, i : \mathtt{I} \vdash R \triangleright \Delta, y : \xi(G \upharpoonright \mathtt{p}_0)$ By inversion
$\Gamma\{\mathrm{n}/i\} \vdash u : \langle G\{\mathrm{n}/i\} \rangle \quad \emptyset; C \vdash G\{\mathrm{n}/i\} \blacktriangleright \mathsf{Type}, C \vdash \mathtt{p}_0, \mathtt{p}_1, p\{\mathrm{n}/i\}$
$C\{\mathrm{n}/i\} \vdash \mathsf{pid}(G\{\mathrm{n}/i\}) = \{\mathtt{p}_0\{\mathrm{n}/i\}, \mathtt{p}_1\{\mathrm{n}/i\}, p\{\mathrm{n}/i\}\}$
$\Gamma\{\mathrm{n}/i\}; C \vdash R\{\mathrm{n}/i\} \triangleright \Delta\{\mathrm{n}/i\}, y : \xi(G\{\mathrm{n}/i\} \upharpoonright \mathtt{p}_0\{\mathrm{n}/i\})$ By i.h.

$\Gamma\{\mathrm{n}/i\}; C \vdash \bar{u}[\mathtt{p}_0\{\mathrm{n}/i\}, \mathtt{p}_1\{\mathrm{n}/i\}, p\{\mathrm{n}/i\}](y).R\{\mathrm{n}/i\} \triangleright \Delta\{\mathrm{n}/i\}$ By rule [TMCast]

The other case is symmetric.

$\Gamma; C, i : \mathtt{I} \vdash k!\langle \mathtt{p}, e\rangle; R \triangleright \Delta, k :!\langle \mathtt{p}, S\rangle; T$ and $\mathrm{n} \leq \max(\mathtt{I})$ By assumption
$\Gamma; C, i : \mathtt{I} \vdash e \triangleright S \quad C, i : \mathtt{I} \vdash \mathtt{p} \quad \Gamma; C, i : \mathtt{I} \vdash R \triangleright \Delta, k : T$ and $\mathrm{n} \leq \max(\mathtt{I})$ By inversion
$\Gamma\{\mathrm{n}/i\}; C \vdash e\{\mathrm{n}/i\} \triangleright S \quad C \vdash \mathtt{p}\{\mathrm{n}/i\} \quad \Gamma\{\mathrm{n}/i\}; C \vdash R\{\mathrm{n}/i\} \triangleright \Delta\{\mathrm{n}/i\}, k : T\{\mathrm{n}/i\}$ By i.h.
$\Gamma\{\mathrm{n}/i\}; C \vdash k!\langle \mathtt{p}\{\mathrm{n}/i\}, e\{\mathrm{n}/i\}\rangle; R\{\mathrm{n}/i\} \triangleright$
$\Delta\{\mathrm{n}/i\}, k :!\langle \mathtt{p}\{\mathrm{n}/i\}, S\rangle; T\{\mathrm{n}/i\}$ By rule [TSend]

Cases: [TRcv], [TSel], [TBr] are symmetric.

$\Gamma; C, i : \mathtt{I} \vdash (\nu a)R \triangleright \Delta$ and $\mathrm{n} \leq \max(\mathtt{I})$ By assumption
$\Gamma, a : U; C, i : \mathtt{I} \vdash R \triangleright \Delta$ and $\mathrm{n} \leq \max(\mathtt{I})$ By inversion
$\Gamma\{\mathrm{n}/i\}, a : U; C \vdash R\{\mathrm{n}/i\} \triangleright \Delta\{\mathrm{n}/i\}$ By i.h.
$\Gamma\{\mathrm{n}/i\}; C \vdash (\nu a)R\{\mathrm{n}/i\} \triangleright \Delta\{\mathrm{n}/i\}$ By rule [Tnu]
where $(\nu a)R\{\mathrm{n}/i\} = (\nu a)R\{\mathrm{n}/i\}$.

$\Gamma; C, i : \mathtt{I} \vdash R \mid S \triangleright \Delta, \Delta'$ and $\mathrm{n} \leq \max(\mathtt{I})$ By assumption
$\Gamma; C, i : \mathtt{I} \vdash R \triangleright \Delta \quad \Gamma; C, i : \mathtt{I} \vdash S \triangleright \Delta'$ and $\mathrm{n} \leq \max(\mathtt{I})$ By inversion
$\Gamma\{\mathrm{n}/i\}; C \vdash R\{\mathrm{n}/i\} \triangleright \Delta\{\mathrm{n}/i\} \quad \Gamma\{\mathrm{n}/i\}; C \vdash S\{\mathrm{n}/i\} \triangleright \Delta'\{\mathrm{n}/i\}$ By i.h.
$\Gamma\{\mathrm{n}/i\}; C \vdash R\{\mathrm{n}/i\} \mid S\{\mathrm{n}/i\} \triangleright \Delta\{\mathrm{n}/i\}, \Delta'\{\mathrm{n}/i\}$ By rule [TConc]
where $R\{\mathrm{n}/i\} \mid S\{\mathrm{n}/i\} = (R \mid S)\{\mathrm{n}/i\}$ and $\Delta\{\mathrm{n}/i\}, \Delta'\{\mathrm{n}/i\} = (\Delta, \Delta')\{\mathrm{n}/i\}$.

Cases [TVar], [TInact], [TIF] are trivial.

$\Gamma; C, i : \mathtt{I} \vdash R \triangleright \Delta'$ and $\mathrm{n} \leq \max(\mathtt{I})$ By assumption
$\Gamma; C, i : \mathtt{I} \vdash R \triangleright \Delta \quad \Delta <: \Delta'$ and $\mathrm{n} \leq \max(\mathtt{I})$ By inversion
$\Gamma\{\mathrm{n}/i\}; C \vdash R\{\mathrm{n}/i\} \triangleright \Delta\{\mathrm{n}/i\} \quad \Delta <: \Delta'$ By i.h.
$\Delta <: \Delta' \Rightarrow \Delta\{\mathrm{n}/i\} <: \Delta'\{\mathrm{n}/i\}$ By Definition 3.4.49
$\Gamma\{\mathrm{n}/i\}; C \vdash R\{\mathrm{n}/i\} \triangleright \Delta'\{\mathrm{n}/i\}$ By rule [$<:$]

$\Gamma; C, i : \mathtt{I} \vdash (\nu s)R \triangleright \Delta$ and $\mathrm{n} \leq \max(\mathtt{I})$ By assumption
$\Gamma; C, i : \mathtt{I} \vdash R \triangleright \Delta, s[\hat{\mathtt{p}}_1] : T_1, ..., s[\hat{\mathtt{p}}_n] : T_\mathrm{n}$ and $\mathrm{n} \leq \max(\mathtt{I})$ By inversion
$\Gamma\{\mathrm{n}/i\}; C \vdash R\{\mathrm{n}/i\} \triangleright \Delta\{\mathrm{n}/i\}, s[\hat{\mathtt{p}}_1] : T_1\{\mathrm{n}/i\}, ..., s[\hat{\mathtt{p}}_n] : T_n\{\mathrm{n}/i\}$ By i.h.
$\Gamma\{\mathrm{n}/i\}; C \vdash (\nu s)R\{\mathrm{n}/i\} \triangleright \Delta\{\mathrm{n}/i\}$ By rule [TCres]
where $(\nu s)R\{\mathrm{n}/i\} = ((\nu s)R)\{\mathrm{n}/i\}$.

$\Gamma, X : \tilde{S}\tilde{T}; C, i : \mathtt{I} \vdash X\langle \tilde{e}, \tilde{k} \rangle \triangleright \Delta, \tilde{k} : \tilde{T}$ and $\mathrm{n} \leq \max(\mathtt{I})$ By assumption
$\Gamma; C, i : \mathtt{I} \vdash \tilde{e} : \tilde{S} \quad \Delta$ end only and $\mathrm{n} \leq \max(\mathtt{I})$ By inversion
$\Gamma\{\mathrm{n}/i\}; C \vdash \tilde{e}\{\mathrm{n}/i\} : \tilde{S} \quad \Delta$ end only By i.h.
$\Gamma\{\mathrm{n}/i\}, X : \tilde{S}\tilde{T}; C \vdash X\langle \tilde{e}\{\mathrm{n}/i\}, \tilde{k} \rangle \triangleright \Delta\{\mathrm{n}/i\}, \tilde{k} : \tilde{T}$ By rule [TVar]

$\Gamma; C, i : \mathtt{I} \vdash \mathsf{def}\ X(\tilde{y}, \tilde{y'}) = R\ \mathsf{in}\ R' \triangleright \Delta$ and $\mathrm{n} \leq \max(\mathtt{I})$ By assumption
$\Gamma, X : \tilde{S}\tilde{T}, \tilde{y} : \tilde{S}; C, i : \mathtt{I} \vdash R \triangleright \tilde{y'} : \tilde{T} \quad \Gamma, X : \tilde{S}\tilde{T}; C, i : \mathtt{I} \vdash R' \triangleright \Delta$
and $\mathrm{n} \leq \max(\mathtt{I})$ By inversion
$\Gamma\{\mathrm{n}/i\}, X : \tilde{S}\tilde{T}\{\mathrm{n}/i\}, \tilde{y} : \tilde{S}; C \vdash R\{\mathrm{n}/i\} \triangleright \tilde{y'} : \tilde{T}\{\mathrm{n}/i\}$
$\Gamma, X : \tilde{S}\tilde{T}\{\mathrm{n}/i\}; C \vdash R'\{\mathrm{n}/i\} \triangleright \Delta\{\mathrm{n}/i\}$ By i.h.
$\Gamma\{\mathrm{n}/i\}; C \vdash \mathsf{def}\ X(\tilde{y}, \tilde{y'}) = R\{\mathrm{n}/i\}\ \mathsf{in}\ R'\{\mathrm{n}/i\} \triangleright \Delta\{\mathrm{n}/i\}$ By rule [TDef]
where $\mathsf{def}\ X(\tilde{y}, \tilde{y'}) = R\{\mathrm{n}/i\}\ \mathsf{in}\ R'\{\mathrm{n}/i\} = (\mathsf{def}\ X(\tilde{y}, \tilde{y'}) = R\ \mathsf{in}\ R')\{\mathrm{n}/i\}$

(3) is by induction on the typing judgement $\Gamma; C \vdash R \triangleright \Delta$. We present the most appealing cases, including those most difficult.

$\Gamma; C \vdash \bar{u}[\mathtt{p}_0, \mathtt{p}_1, p](y).R \triangleright \Delta, y' : T'$ By assumption
$\Gamma \vdash u : \langle G \rangle \quad \emptyset; C \vdash G \blacktriangleright \mathsf{Type}, C \vdash \mathtt{p}_0, \mathtt{p}_1, p \quad C \vdash \mathsf{pid}(G) = \{\mathtt{p}_0, \mathtt{p}_1, p\}$,
$\Gamma; C \vdash R \triangleright \Delta, y' : T', y : \xi(G \upharpoonright \mathtt{p}_0)$ By inversion
$\Gamma \vdash u : \langle G \rangle \quad \emptyset; C \vdash G \blacktriangleright \mathsf{Type}, C \vdash \mathtt{p}_0, \mathtt{p}_1, p \quad C \vdash \mathsf{pid}(G) = \{\mathtt{p}_0, \mathtt{p}_1, p\}$
$\Gamma; C \vdash R\{s[\hat{\mathtt{p}}]/y'\} \triangleright \Delta, s[\hat{\mathtt{p}}] : T', y : \xi(G \upharpoonright \mathtt{p}_0)$ By i.h.
$\Gamma; C \vdash \bar{u}[\mathtt{p}_0, \mathtt{p}_1, p](y).R\{s[\hat{\mathtt{p}}]/y'\} \triangleright \Delta, s[\hat{\mathtt{p}}] : T'$ By rule [TMCast]

The other case is symmetric.

$\Gamma; C \vdash k!\langle \mathtt{p}, e \rangle; R \triangleright \Delta, k :!\langle \mathtt{p}, S \rangle; T, y : T'$ By assumption
$\Gamma; C \vdash e \triangleright S \quad C \vdash \mathtt{p} \quad \Gamma; C \vdash R \triangleright \Delta, y : T', k : T$ By inversion
$\Gamma; C \vdash e \triangleright S \quad C \vdash \mathtt{p} \quad \Gamma; C \vdash R\{s[\hat{\mathtt{p}}]/y\} \triangleright \Delta, s[\hat{\mathtt{p}}] : T', k : T$ By i.h.

$\Gamma; C \vdash k!\langle \mathtt{p}, e\rangle; R\{\hat{\mathtt{p}}/y\} \triangleright \Delta, s[\hat{\mathtt{p}}] : T', k :!\langle \mathtt{p}, S\rangle; T$ By rule [TSend]

$\Gamma; C \vdash y!\langle \mathtt{p}, e\rangle; R \triangleright \Delta, y :!\langle \mathtt{p}, S\rangle; T$ By assumption
$\Gamma; C \vdash e \triangleright S \quad C \vdash \mathtt{p} \quad \Gamma; C \vdash R \triangleright \Delta, y : T$ By inversion
$\Gamma; C \vdash e \triangleright S \quad C \vdash \mathtt{p} \quad \Gamma; C \vdash R\{s[\hat{\mathtt{p}}]/y\} \triangleright \Delta, s[\hat{\mathtt{p}}] : T$ By i.h.
$\Gamma; C \vdash y\{s[\hat{\mathtt{p}}]/y\}!\langle \mathtt{p}, e\rangle; R\{s[\hat{\mathtt{p}}]/y\} \triangleright \Delta, s[\hat{\mathtt{p}}] :!\langle \mathtt{p}, S\rangle; T$ By rule [TSend]

Cases: [TRcv], [TSel], [TBr] are symmetric.

$\Gamma; C \vdash (\nu a)R \triangleright \Delta, y : T$ By assumption
$\Gamma, a : U; C \vdash R \triangleright \Delta, y : T$ By inversion
$\Gamma, a : U; C \vdash R \triangleright \Delta, s[\hat{\mathtt{p}}] : T$ By i.h.
$\Gamma; C \vdash (\nu a)R \triangleright \Delta, s[\hat{\mathtt{p}}] : T$ By rule [Tnu]

$\Gamma; C \vdash R \mid S \triangleright \Delta, \Delta', y : T$ By assumption
$\Gamma; C \vdash R \triangleright \Delta, y : T \quad \Gamma; C \vdash S \triangleright \Delta'$ By inversion
$\Gamma; C \vdash R\{s[\hat{\mathtt{p}}]/y\} \triangleright \Delta, s[\hat{\mathtt{p}}] : T \quad \Gamma; C \vdash S \triangleright \Delta'$ By i.h.
$\Gamma; C \vdash R\{s[\hat{\mathtt{p}}]/y\} \mid S \triangleright \Delta, \Delta', s[\hat{\mathtt{p}}] : T$ By rule [TConc]

The other case is symmetric. Cases [TVar], [TInact], [TIF] are trivial.

$\Gamma; C \vdash R \triangleright \Delta', y : T'$ By assumption
$\Gamma; C \vdash R \triangleright \Delta, y : T \quad \Delta, y : T <: \Delta', y : T'$ where $T <: T'$ By inversion
$\Gamma; C \vdash R\{s[\hat{\mathtt{p}}]/y\} \triangleright \Delta, y : T \quad \Delta, y : T <: \Delta', y : T'$ By i.h.
$\Delta, y : T <: \Delta', y : T' \Rightarrow \Delta, s[\hat{\mathtt{p}}] : T <: \Delta', s[\hat{\mathtt{p}}] : T'$ By i.h.
$\Gamma; C \vdash R\{s[\hat{\mathtt{p}}]/y\} \triangleright \Delta', s[\hat{\mathtt{p}}] : T'$ By rule [$<:$]

$\Gamma; C \vdash (\nu s)R \triangleright \Delta, y : T$ By assumption
$\Gamma; C \vdash R \triangleright \Delta, y : T, s[\mathtt{p}_1] : T_1, ..., s[\mathtt{p}_n] : T_n$ By inversion
$\Gamma; C \vdash R\{s'[\hat{\mathtt{p}}]/y\} \triangleright \Delta, s'[\hat{\mathtt{p}}] : T, s[\mathtt{p}_1] : T_1\{\mathrm{n}/i\}, ..., s[\mathtt{p}_n] : T_n\{\mathrm{n}/i\}$ By i.h.
$\Gamma; C \vdash (\nu s)R\{s'[\hat{\mathtt{p}}]/y\} \triangleright \Delta, s'[\hat{\mathtt{p}}] : T$ By rule [TCres]
where $(\nu s)R\{s'[\hat{\mathtt{p}}]/y\} = ((\nu s)R)\{s'[\hat{\mathtt{p}}]/y\}$.

$\Gamma, X : \tilde{S}\tilde{T}; C \vdash X\langle \tilde{e}, \tilde{k}\rangle \triangleright \Delta, \tilde{k} : \tilde{T}, y : \mathsf{end}$ By assumption
$\Gamma; C \vdash \tilde{e} : \tilde{S} \quad \Delta$ end only By inversion
$\Gamma; C \vdash \tilde{e} : \tilde{S} \quad \Delta$ end only By i.h.
$\Gamma, X : \tilde{S}\tilde{T}; C \vdash X\langle \tilde{e}, \tilde{k}\rangle \triangleright \Delta, \tilde{k} : \tilde{T}$ By rule [TVar]
$\Gamma, X : \tilde{S}\tilde{T}; C \vdash X\langle \tilde{e}, \tilde{k}\rangle \triangleright \Delta, \tilde{k} : \tilde{T}, s[\hat{\mathtt{p}}] : \mathsf{end}$ By rule Lemma 3.4.52(2)

$\Gamma, X : \tilde{S}\tilde{T}; C \vdash X\langle \tilde{e}, y\rangle \triangleright \Delta, y : T$ By assumption
$\Gamma; C \vdash \tilde{e} : \tilde{S} \quad \Delta$ end only By inversion
$\Gamma, X : \tilde{S}T; C \vdash X\langle \tilde{e}, y\{s[\hat{\mathtt{p}}]/y\}\rangle \triangleright \Delta, s[\hat{\mathtt{p}}] : T$ By rule [TVar]

$\Gamma; C \vdash \mathsf{def}\ X(\tilde{y}, \tilde{y'}) = R\ \mathsf{in}\ R' \triangleright \Delta, y'' : T'$ By assumption
$\Gamma, X : \tilde{S}\tilde{T}, \tilde{y} : \tilde{S}; C \vdash R \triangleright \tilde{y'} : \tilde{T} \quad \Gamma, X : \tilde{S}\tilde{T}; C \vdash R' \triangleright \Delta, y'' : T'$ By inversion
$\Gamma, X : \tilde{S}\tilde{T}, \tilde{y} : \tilde{S}; C \vdash R \triangleright \tilde{y'} : \tilde{T} \quad \Gamma, X : \tilde{S}\tilde{T}; C \vdash R'\{s[\hat{\mathtt{p}}]/y''\} \triangleright \Delta, s[\hat{\mathtt{p}}] : T'$ By i.h.
$\Gamma; C \vdash \mathsf{def}\ X(\tilde{y}, \tilde{y'}) = R\ \mathsf{in}\ R'\{s[\hat{\mathtt{p}}]/y''\} \triangleright \Delta, s[\hat{\mathtt{p}}] : T'$ By rule [TDef]

(4) is by induction on the typing judgement $\Gamma; C \vdash R \triangleright \Delta$. We present the most appealing cases, including those most difficult.

$\Gamma, x : S; C \vdash \bar{u}[\mathtt{p}_0, \mathtt{p}_1, p](y).R \triangleright \Delta$ and $\Gamma; C \vdash v : S$ By assumption
$\Gamma, x : S \vdash u : \langle G\rangle \quad \emptyset; C \vdash G \blacktriangleright \mathsf{Type}, C \vdash \mathtt{p}_0, \mathtt{p}_1, p \quad C \vdash \mathsf{pid}(G) = \{\mathtt{p}_0, \mathtt{p}_1, p\}$,
$\Gamma, x : S; C \vdash R \triangleright \Delta, y' : T', y : \xi(G \upharpoonright \mathtt{p}_0)$ and $\Gamma; C \vdash v : S$ By inversion
$\Gamma \vdash u : \langle G\rangle \quad \emptyset; C \vdash G \blacktriangleright \mathsf{Type}, C \vdash \mathtt{p}_0, \mathtt{p}_1, p$ By Strengthening
$C \vdash \mathsf{pid}(G) = \{\mathtt{p}_0, \mathtt{p}_1, p\} \quad \Gamma; C \vdash R\{v/x\} \triangleright \Delta y : \xi(G \upharpoonright \mathtt{p}_0)$ By i.h.
$\Gamma; C \vdash \bar{u}[\mathtt{p}_0, \mathtt{p}_1, p](y).R\{v/x\} \triangleright \Delta$ By rule [TMCast]

The other case is symmetric.

$\Gamma, x : S; C \vdash k!\langle \mathtt{p}, e\rangle; R \triangleright \Delta, k : !\langle \mathtt{p}, S\rangle; T, y : T'$ and $\Gamma; C \vdash v : S$ By assumption
$\Gamma, x : S; C \vdash e \triangleright S' \quad C \vdash \mathtt{p} \quad \Gamma, x : S; C \vdash R \triangleright \Delta, k : T$ By inversion
$\Gamma; C \vdash e\{v/x\} \triangleright S \quad C \vdash \mathtt{p} \quad \Gamma; C \vdash R\{v/x\} \triangleright \Delta, k : T$ By i.h.
$\Gamma; C \vdash k!\langle \mathtt{p}, e\{v/x\}\rangle; R\{v/x\} \triangleright \Delta, k : !\langle \mathtt{p}, S\rangle; T$ By rule [TSend]

$\Gamma, x : S; C \vdash k?\langle \mathtt{p}, x' \rangle; R \triangleright \Delta, k :?\langle \mathtt{p}, S' \rangle; T$ and $\Gamma; C \vdash v : S$ — By assumption
$C \vdash \mathtt{p} \quad \Gamma, x : S, x' : S'; C \vdash R \triangleright \Delta, k : T$ — By inversion
$C \vdash \mathtt{p} \quad \Gamma, x' : S'; C \vdash R\{v/x\} \triangleright \Delta, s[\hat{\mathtt{p}}] : T$ — By i.h.
$\Gamma; C \vdash k?\langle \mathtt{p}, x' \rangle; R\{v/x\} \triangleright \Delta, s[\hat{\mathtt{p}}] :?\langle \mathtt{p}, S' \rangle; T$ — By rule [TRcv]

Cases: [TSel], [TBr] are symmetric.

$\Gamma, x : S; C \vdash (\nu a)R \triangleright \Delta$ and $\Gamma; C \vdash v : S$ — By assumption
$\Gamma, x : S, a : \langle G \rangle; C \vdash R \triangleright \Delta$ and $\Gamma; C \vdash v : S$ — By inversion
$\Gamma, a : \langle G \rangle; C \vdash R\{v/x\} \triangleright \Delta$ — By i.h.
$\Gamma; C \vdash (\nu a)R\{v/x\} \triangleright \Delta$ — By rule [Tnu]

$\Gamma, x : S; C \vdash R \mid S \triangleright \Delta, \Delta'$ and $\Gamma; C \vdash v : S$ — By assumption
$\Gamma, x : S; C \vdash R \triangleright \Delta \quad \Gamma, x : S; C \vdash S \triangleright \Delta'$ and $\Gamma; C \vdash v : S$ — By inversion
$\Gamma; C \vdash R\{v/x\} \triangleright \Delta \quad \Gamma; C \vdash S\{v/x\} \triangleright \Delta'$ — By i.h.
$\Gamma; C \vdash R\{v/x\} \mid S\{v/x\} \triangleright \Delta, \Delta'$ — By rule [TConc]

The other case is symmetric. Cases [TVar], [TInact], [TIF] are trivial.

$\Gamma, x : S; C \vdash R \triangleright \Delta'$ and $\Gamma; C \vdash v : S$ — By assumption
$\Gamma, x : S; C \vdash R \triangleright \Delta \quad \Delta <: \Delta'$ and $\Gamma; C \vdash v : S$ — By inversion
$\Gamma; C \vdash R\{v/x\} \triangleright \Delta \quad \Delta <: \Delta'$ — By i.h.
$\Gamma; C \vdash R\{v/x\} \triangleright \Delta'$ — By rule [$<:$]

$\Gamma, x : S; C \vdash (\nu s)R \triangleright \Delta$ — By assumption
$\Gamma, x : S; C \vdash R \triangleright \Delta, s[\hat{\mathtt{p}}_1] : T_1, ..., s[\hat{\mathtt{p}}_n] : T_{\mathrm{n}}$ — By inversion
$\Gamma; C \vdash R\{v/x\} \triangleright \Delta, s[\hat{\mathtt{p}}_1] : T_1, ..., s[\hat{\mathtt{p}}_n] : T_{\mathrm{n}}$ — By i.h.
$\Gamma; C \vdash (\nu s)R\{v/x\} \triangleright \Delta$ — By rule [TCres]

$\Gamma, X : \tilde{S}\tilde{T}, x : S'; C \vdash X\langle \tilde{e}, \tilde{k} \rangle \triangleright \Delta, \tilde{k} : \tilde{T}$ and $\Gamma; C \vdash v : S$ — By assumption
$\Gamma, x : S'; C \vdash \tilde{e} : \tilde{S} \quad \Delta$ end only and $\Gamma; C \vdash v : S$ — By inversion
$\Gamma; C \vdash \tilde{e}\{v/x\} : \tilde{S} \quad \Delta$ end only — By i.h.

$\Gamma, X : \tilde{S}\tilde{T}; C \vdash X\langle \tilde{e}\{v/x\}, \tilde{k}\rangle \triangleright \Delta, \tilde{k} : \tilde{T}$ By rule [TVar]

$\Gamma, x : S'; C \vdash \mathsf{def}\ X(\tilde{y}, \tilde{y'}) = R\ \mathsf{in}\ R' \triangleright \Delta$ and $\Gamma; C \vdash v : S$ By assumption
$\Gamma, X : \tilde{S}\tilde{T}, \tilde{y} : \tilde{S}, x : S'; C \vdash R \triangleright \tilde{y'} : \tilde{T} \quad \Gamma, X : \tilde{S}\tilde{T}, x : S'; C \vdash R' \triangleright \Delta$
and $\Gamma; C \vdash v : S$ By inversion
$\Gamma, X : \tilde{S}\tilde{T}, \tilde{y} : \tilde{S}; C \vdash R\{v/x\} \triangleright \tilde{y'} : \tilde{T} \quad \Gamma, X : \tilde{S}\tilde{T}; C \vdash R'\{v/x\} \triangleright \Delta$ By i.h.
$\Gamma; C \vdash \mathsf{def}\ X(\tilde{y}, \tilde{y'}) = R\{v/x\}\ \mathsf{in}\ R'\{v/x\} \triangleright \Delta$ By rule [TDef]

□

Theorem 3.4.57 (Type Preservation)**.** If $\Gamma; C \vdash E \triangleright \tau$, and $E \rightarrow^\star E'$, then there exists τ' where $\tau \Rightarrow \tau'$, such that $\Gamma; C \vdash E' \triangleright \tau'$.

Proof. By induction over the derivation of $E \rightarrow^\star E'$.

Case

$$(\lambda n.E)\ \mathrm{n} \longrightarrow E\{\mathrm{n}/n\}$$

$\Gamma; C \vdash (\lambda n.E)\ \mathrm{n} \triangleright \tau$ By assumption
$\Gamma; C \vdash \lambda n.E \triangleright \Pi n : \mathtt{T}.\tau$ and $C \vdash \mathrm{n} \geq \min(\mathtt{T})$ By inversion
$\Gamma; C, n : \mathtt{T} \vdash E \triangleright \tau$ and $C \vdash \mathrm{n} \geq \min(\mathtt{T})$ By inversion
$\Gamma; C\{\mathrm{n}/n\} \vdash E\{\mathrm{n}/n\} \triangleright \tau\{\mathrm{n}/n\}$ By substitution lemma (1)

Case

$$\mathbf{R}\ S\ \lambda i.\lambda X.R\ 0 \longrightarrow S$$

$\Gamma; C \vdash \mathbf{R}\ S\ \lambda i.\lambda X.R\ 0 \triangleright \Delta\ 0$ By assumption
$\Gamma; C \vdash \mathbf{R}\ S\ \lambda i.\lambda X.R \triangleright \Pi i : \mathtt{I}.\Delta$ and $C \vdash 0$ By inversion
$\Gamma; C \vdash S \triangleright \Delta\ 0$ By inversion

Case

$$\mathbf{R}\ S\ \lambda i.\lambda X.R\ \mathrm{n}+1 \longrightarrow R\{\mathrm{n}/i\}\{\mathbf{R}\ S\ \lambda i.\lambda X.R\ \mathrm{n}/X\}$$

$\Gamma; C \vdash \mathbf{R}\ S\ \lambda i.\lambda X.R\ \mathrm{n}+1 \rhd \Delta\ \mathrm{n}+1$ By assumption
$\Gamma; C \vdash \mathbf{R}\ S\ \lambda i.\lambda X.R \rhd \Pi i : \mathtt{I}.\Delta$ and $C \vdash \mathrm{n}+1 = \max(\mathtt{I})$ By inversion
$\Gamma; C \vdash S \rhd \Delta\ 0$ and $\Gamma, X : \Delta\ i; C, i : \mathtt{I} \vdash R \rhd \Delta\ i+1$ By inversion
$\Gamma, X : \Delta\ \mathrm{n}; C \vdash R\{\mathrm{n}/i\} \rhd \Delta \mathrm{n}+1$ By substitution lemma (2)
$\Gamma; C \vdash \mathbf{R}\ S\ \lambda i.\lambda X.R\ \mathrm{n} \rhd \Delta\ \mathrm{n}$ By i.h.
$\Gamma; C \vdash R\{\mathrm{n}/i\}\{\mathbf{R}\ S\ \lambda i.\lambda X.R\ \mathrm{n}/X\} \rhd \Delta \mathrm{n}+1$ By substitution lemma (3)

Case

$$\bar{a}[\hat{\mathtt{p}}_0..\hat{\mathtt{p}}_\mathrm{n}](y_0).R_0 \mid a[\hat{\mathtt{p}}_1](y_1).R_1 \mid ...a[\hat{\mathtt{p}}_\mathrm{n}](y_\mathrm{n}).R_\mathrm{n} \longrightarrow (\nu s)(R_0\{s[\hat{\mathtt{p}}_0]/y_0\} \mid ... \mid R_\mathrm{n}\{s[\hat{\mathtt{p}}_\mathrm{n}]/y_\mathrm{n}\} \mid s : \emptyset)$$

$\Gamma; C \vdash \bar{a}[\hat{\mathtt{p}}_0..\hat{\mathtt{p}}_\mathrm{n}](y_0).R_0 \mid a[\hat{\mathtt{p}}_1](y_1).R_1 \mid ...a[\hat{\mathtt{p}}_\mathrm{n}](y_\mathrm{n}).R_\mathrm{n} \rhd \Delta$ By assumption
$\Gamma; C \vdash \bar{a}[\hat{\mathtt{p}}_0..\hat{\mathtt{p}}_\mathrm{n}](y_0).R_0 \rhd \Delta_1$, ..., $\Gamma; C \vdash a[\hat{\mathtt{p}}_\mathrm{n}](y_\mathrm{n}).R_\mathrm{n} \rhd \Delta_{\mathrm{n}+1}$,
where $\Delta = \Delta_1, ..., \Delta_{n+1}$ By inversion
$\Gamma; C \vdash R_0 \rhd \Delta_1, y : \xi(G \upharpoonright \hat{\mathtt{p}}_0)$ By inversion
$\Gamma; C \vdash R_0\{s[\hat{\mathtt{p}}_0]/y_0\} \rhd \Delta_1, s[\hat{\mathtt{p}}_0] : \xi(G \upharpoonright \hat{\mathtt{p}}_0)$ By substitution lemma (4)
...
$\Gamma; C \vdash R_\mathrm{n}\{s[\hat{\mathtt{p}}_\mathrm{n}]/y_\mathrm{n}\} \rhd \Delta_{\mathrm{n}+1}, s[\hat{\mathtt{p}}_\mathrm{n}] : \xi(G \upharpoonright \hat{\mathtt{p}}_\mathrm{n})$ By substitution lemma (4)
$\Gamma; C \vdash R_0\{s[\hat{\mathtt{p}}_0]/y_0\} \mid ... \mid R_\mathrm{n}\{s[\hat{\mathtt{p}}_\mathrm{n}]/y_\mathrm{n}\} \rhd$
$\Delta_1, ..., \Delta_{\mathrm{n}+1}, s[\hat{\mathtt{p}}_0] : \xi(G \upharpoonright \hat{\mathtt{p}}_0), ..., s[\hat{\mathtt{p}}_\mathrm{n}] : \xi(G \upharpoonright \hat{\mathtt{p}}_\mathrm{n})$ By rule [Conc]
$\Gamma; C \vdash (\nu s)(R_0\{s[\hat{\mathtt{p}}_0]/y_0\} \mid ... \mid R_\mathrm{n}\{s[\hat{\mathtt{p}}_\mathrm{n}]/y_\mathrm{n}\} \mid s : \emptyset) \rhd \Delta$ By rule [Cres]

Case

$$s[\hat{\mathtt{p}}]!\langle\hat{\mathtt{q}}, v\rangle; R \mid s : h \longrightarrow R \mid s : h \cdot (\hat{\mathtt{p}}, \hat{\mathtt{q}}, v)$$

$\Gamma; C \vdash s[\hat{\mathtt{p}}]!\langle\hat{\mathtt{q}}, v\rangle; R \mid s : h \rhd \Delta$ By assumption
$\Gamma; C \vdash s[\hat{\mathtt{p}}]!\langle\hat{\mathtt{q}}, v\rangle; R \rhd \Delta_1$, $\Gamma; C \vdash s : h \rhd \Delta_2$ where $\Delta = \Delta_1 \star \Delta_2$ By inversion
$\Delta_1 = \Delta_1', s[\hat{\mathtt{p}}] :!\langle\hat{\mathtt{q}}, S\rangle; T$ By rule [TSend]
$\Gamma; C \vdash R \rhd \Delta_1', s[\hat{\mathtt{p}}] : T$ By inversion
$\Gamma; C \vdash s : h \cdot (\hat{\mathtt{p}}, \hat{\mathtt{q}}, v) \rhd \Delta_2; \{s[\hat{\mathtt{p}}] :!\langle\hat{\mathtt{q}}, S\rangle\}$ By rule [TRSend]

$\Gamma; C \vdash R \mid s : h \cdot (\hat{\mathtt{p}}, \hat{\mathtt{q}}, v) \rhd \Delta_1', s[\hat{\mathtt{p}}] : T \star \Delta_2; \{s[\hat{\mathtt{p}}] :!\langle \hat{\mathtt{q}}, S\rangle\}$ By rule [TRConc]

Case

$$s[\hat{\mathtt{p}}]?(\hat{\mathtt{q}}, x); R \mid s : (\hat{\mathtt{q}}, \hat{\mathtt{p}}, v) \cdot h \longrightarrow R\{v/x\} \mid s : h$$

$\Gamma; C \vdash s[\hat{\mathtt{p}}]?(\hat{\mathtt{q}}, x); R \mid s : (\hat{\mathtt{q}}, \hat{\mathtt{p}}, v) \cdot h \rhd \Delta$ By assumption
$\Gamma; C \vdash s[\hat{\mathtt{p}}]?(\hat{\mathtt{q}}, x); R \rhd \Delta_1$, $\Gamma; C \vdash s : (\hat{\mathtt{q}}, \hat{\mathtt{p}}, v) \cdot h \rhd \Delta_2$ where $\Delta = \Delta_1 \star \Delta_2$ By inversion
$\Delta_1 = \Delta_1', s[\hat{\mathtt{p}}] :?\langle \hat{\mathtt{q}}, S\rangle; T$ By rule [TRecv]
$\Gamma, x : S; C \vdash R \rhd \Delta_1', s[\hat{\mathtt{p}}] : T$ By inversion
$\Delta_2 = \{s[\hat{\mathtt{p}}] :!\langle \hat{\mathtt{q}}, S\rangle\}; \Delta_2'$ By rule [TRSend]
$\Gamma; C \vdash R\{v/x\} \rhd \Delta_1', s[\hat{\mathtt{p}}] : T$ By substitution lemma (5)
$\Gamma; C \vdash R\{v/x\} \mid s : h \rhd \Delta_1', s[\hat{\mathtt{p}}] : T \star \Delta_2'$ By rule [TRConc]
$\Delta_1', s[\hat{\mathtt{p}}] : T \star \{s[\hat{\mathtt{p}}] :!\langle \hat{\mathtt{q}}, S\rangle\}; \Delta_2' = \Delta_1', \Delta_2, s[\hat{\mathtt{p}}] : T$

Case

$$s[\hat{\mathtt{p}}] \oplus \langle \hat{\mathtt{q}}, l\rangle; R \mid s : h \longrightarrow R \mid s : h \cdot (\hat{\mathtt{p}}, \hat{\mathtt{q}}, l)$$

$\Gamma; C \vdash s[\hat{\mathtt{p}}] \oplus \langle \hat{\mathtt{q}}, l\rangle; R \mid s : h \rhd \Delta$ By assumption
$\Gamma; C \vdash s[\hat{\mathtt{p}}] \oplus \langle \hat{\mathtt{q}}, l\rangle; R \rhd \Delta_1$, $\Gamma; C \vdash s : h \rhd \Delta_2$ where $\Delta = \Delta_1 \star \Delta_2$ By inversion
$\Delta_1 = \Delta_1', s[\hat{\mathtt{p}}] : \oplus\langle \hat{\mathtt{q}}, \{l_i : T_i\}_{i \in I}\rangle$ By rule [TSel]
$\Gamma; C \vdash R \rhd \Delta_1', s[\hat{\mathtt{p}}] : T_i, l = l_i$ By inversion
$\Gamma; C \vdash s : h \cdot (\hat{\mathtt{p}}, \hat{\mathtt{q}}, l) \rhd \Delta_2; \{s[\hat{\mathtt{p}}] : \oplus\langle \hat{\mathtt{q}}, l\rangle\}$ By rule [TRSel]
$\Gamma; C \vdash R \mid s : h \cdot (\hat{\mathtt{p}}, \hat{\mathtt{q}}, l) \rhd \Delta_1', s[\hat{\mathtt{p}}] : T_i \star \Delta_2; \{s[\hat{\mathtt{p}}] : \oplus\langle \hat{\mathtt{q}}, l\rangle\}$ By rule [TRConc]
$\Delta_1', s[\hat{\mathtt{p}}] : T_i \star \Delta_2; \{s[\hat{\mathtt{p}}] : \oplus\langle \hat{\mathtt{q}}, l\rangle\} = \Delta_1', \Delta_2, s[\hat{\mathtt{p}}] : \oplus\langle \hat{\mathtt{q}}, l : T_i\rangle$
$\Gamma; C \vdash R \mid s : h \cdot (\hat{\mathtt{p}}, \hat{\mathtt{q}}, l) \rhd \Delta_1', \Delta_2, s[\hat{\mathtt{p}}] : \oplus\langle \hat{\mathtt{p}}, \{l_i : T_i\}_{i \in I}\rangle$ By rule [<:]

Case

$$s[\hat{\mathtt{p}}]\&(\hat{\mathtt{q}}, \{l_i : R_i\}_{i \in I}) \mid s : (\hat{\mathtt{q}}, \hat{\mathtt{p}}, l_{i_0}) \cdot h \longrightarrow R_{i_0} \mid s : h \quad (i_0 \in I)$$

$\Gamma; C \vdash s[\hat{\mathtt{p}}]\&(\hat{\mathtt{q}}, \{l_i : R_i\}_{i \in I}) \mid s : (\hat{\mathtt{q}}, \hat{\mathtt{p}}, l_{i_0}) \cdot h \rhd \Delta$ By assumption
$\Gamma; C \vdash s[\hat{\mathtt{p}}]\&(\hat{\mathtt{q}}, \{l_i : R_i\}_{i \in I}) \rhd \Delta_1$, $\Gamma; C \vdash s : (\hat{\mathtt{q}}, \hat{\mathtt{p}}, l_{i_0}) \cdot h \rhd \Delta_2$
where $\Delta = \Delta_1 \star \Delta_2$ By inversion

$\Delta_1 = \Delta_1', s[\hat{\mathtt{p}}] : \&\langle \mathtt{p}, \{l_i : T_k\}_{i\in I}\rangle$ By rule [TBr]
$\forall i \in I.\Gamma; C \vdash R_i \rhd \Delta_1', s[\hat{\mathtt{p}}] : T_i$ By inversion
$\Delta_2 = \{s[\hat{\mathtt{p}}] : \oplus\langle \hat{\mathtt{q}}, l_{i_0}\rangle\}; \Delta_2'$ By rule [TRSel]
$\Gamma; C \vdash R_{i_0} \mid s : h \rhd \Delta_1', s[\hat{\mathtt{p}}] : T_{i_0} \star \{s[\hat{\mathtt{p}}] : \oplus\langle \hat{\mathtt{q}}, l_{i_0}\rangle\}; \Delta_2'$ By rule [TRConc]
$\Delta_1', s[\hat{\mathtt{p}}] : T_{i_0} \star \{s[\hat{\mathtt{p}}] : \oplus\langle \hat{\mathtt{q}}, l_{i_0}\rangle\}; \Delta_2' = \Delta_1', \Delta_2, s[\hat{\mathtt{p}}] : \&\langle \hat{\mathtt{q}}, l_{i_0} : T_{i_0}\rangle$
$\Gamma; C \vdash R_{i_0} \mid s : h \rhd \Delta_1', \Delta_2, s[\hat{\mathtt{p}}] : \&\langle \mathtt{p}, \{l_i : T_k\}_{i\in I}\rangle$ By rule [<:]

Case [IfT, IfF] are trivial.

Case

$$\mathsf{def}\, X(\tilde{y}\tilde{y}') = R' \mathsf{in}\ (\mathbb{X}\langle \tilde{v}\tilde{s}[\hat{\mathtt{p}}]\rangle \mid R) \longrightarrow \mathsf{def}\, X(\tilde{y}\tilde{y}') = R' \mathsf{in}\ (R'\{\tilde{v}/\tilde{y}\}\{\tilde{s}[\hat{\mathtt{p}}]/\tilde{y}'\} \mid R)$$

$\Gamma; C \vdash \mathsf{def}\ X(\tilde{y}\tilde{y}') = S\ \mathsf{in}\ (\mathbb{X}\langle \tilde{v}\tilde{s}[\hat{\mathtt{p}}]\rangle \mid R) \rhd \Delta$ By assumption
$\Gamma, X : \tilde{S}\tilde{T}, \tilde{y} : \tilde{S}; C \vdash R' \rhd \tilde{y}' : \tilde{T} \quad \Gamma, X : \tilde{S}\tilde{T}; C \vdash \mathbb{X}\langle \tilde{v}\tilde{s}[\hat{\mathtt{p}}]\rangle \mid R \rhd \Delta$ By inversion on rule [Def]
$\Gamma, X : \tilde{S}'\tilde{T}; C \vdash X\langle \tilde{v}\tilde{s}[\hat{\mathtt{p}}]\rangle \rhd \Delta_1$ and $\Gamma, X : \tilde{S}'\tilde{T} \vdash R \rhd \Delta_2$
where $\Delta = \Delta_1, \Delta_2$ By inversion on rule [Conc]
$\Gamma; C \vdash \tilde{v} \rhd \tilde{S}$ and $\Delta_1 = \Delta_1', \tilde{s}[\hat{\mathtt{p}}] : \tilde{T}$ By inversion on rule [Var]
$\Gamma, X : \tilde{S}\tilde{T}; C \vdash R'[\tilde{v}/\tilde{y}] \rhd s[\tilde{\hat{\mathtt{p}}}] : \tilde{T}$ By Lemma 3.4.56.5 and Lemma 3.4.56.4
$\Gamma, X : \tilde{S}\tilde{T}; C \vdash R'[\tilde{v}/\tilde{y}] \rhd \Delta_1', s[\tilde{\hat{\mathtt{p}}}] : \tilde{T}$ By Lemma 3.4.52.2
$\Gamma, X : \tilde{S}\tilde{T}; C \vdash R'[\tilde{v}/\tilde{y}] \mid R \rhd \Delta_1', s[\tilde{\hat{\mathtt{p}}}] : \tilde{T}, \Delta_2 = \Delta$ By rule [Conc]
$\Gamma; C \vdash \mathsf{def}\, X(\tilde{y}\tilde{y}') = R' \mathsf{in}\ (R'\{\tilde{v}/\tilde{y}\}\{\tilde{s}[\hat{\mathtt{p}}]/\tilde{y}'\} \mid R) \rhd \Delta$ By rule [Def]

Case

$$R \longrightarrow R' \quad \Rightarrow \quad R \mid S \longrightarrow R' \mid S$$

$\Gamma; C \vdash R \mid S \rhd \Delta$ By assumption
$\Gamma; C \vdash R \rhd \Delta_1$ and $\Gamma; C \vdash S \rhd \Delta_2$ where $\Delta = \Delta_1, \Delta_2$ By rule [Conc]
$\Gamma; C \vdash R' \rhd \Delta_1'$ where $\Delta_1 = \Delta_1'$ or $\Delta_1 \ \rightarrow\ \Delta_1'$ By induction
when $\Delta_1 = \Delta_1'$ then the proof is trivial so we investigate the second case
when $\Delta_1 \ \rightarrow\ \Delta_1'$
$\Delta_1, \Delta_2 \ \rightarrow\ \Delta_1', \Delta_2$ By rule [TR-Context]

$\Gamma \vdash R' \mid S \triangleright \Delta_1', \Delta_2$ By rule [Conc]

Case: [Scop, DefIn] is trivial by induction.

Case

$$R \equiv R' \text{ and } R' \longrightarrow S' \text{ and } S \equiv S' \quad \Rightarrow \quad R \longrightarrow S$$

$\Gamma \vdash R \triangleright \Delta$, $R \equiv R'$, $R' \ \rightarrow \ S'$ and $S' \equiv S$ By assumption
$\Gamma \vdash R' \triangleright \Delta$, $R' \ \rightarrow \ S'$ and $S' \equiv S$ By Theorem 3.4.54
$\Gamma \vdash S' \triangleright \Delta$, $S' \equiv S$ By induction
$\Gamma \vdash S \triangleright \Delta$ By Theorem 3.4.54

□

We say:

- A prefix is *at* $s[\hat{\mathtt{p}}]$ (resp. *at* a) if its subject (i.e. its initial channel) is $s[\hat{\mathtt{p}}]$ (resp. a). Further a prefix is *emitting* if it is request, output, delegation or selection, otherwise it is *receiving*.
- A prefix is *active* if it is not under a prefix, an `if` branch or $\mathbf{R}$ operator, after any unfoldings by [Def]. We write $R\langle s[\hat{\mathtt{p}}]\rangle$ if R contains an active subject at $s[\hat{\mathtt{p}}]$ after applying [Def], and $R\langle s[\hat{\mathtt{p}}]!\rangle$ (resp. $R\langle s[\hat{\mathtt{p}}]?\rangle$) if R contains an emitting (resp. receiving) active prefix at $s[\hat{\mathtt{p}}]$.
- R has a *redex* if (1) it has an active emitting prefix in a process at $s[\hat{\mathtt{p}}]$ and (2) it has a message in the queue at $(s[\hat{\mathtt{q}}], s[\hat{\mathtt{p}}])$ and an active receiving prefix in a process at $s[\hat{\mathtt{p}}]$ among its redexes.

Definition 3.4.58 (Reduction context)**.** Below, we define the reduction context $\mathcal{F}$:

$$\mathcal{F} ::= [_] \mid \mathcal{F} \mid R \mid R \mid \mathcal{F} \mid (\nu w)\mathcal{F} \mid \mathsf{def}\ D\ \mathsf{in}\ \mathcal{F}$$

The definition and proof of communication-safety below is in the style of Yoshida and Vasconcelos (2007); Vasconcelos et al. (2004).

Definition 3.4.59. Process R is *error* if $R \equiv \mathsf{def}\ D\ \mathsf{in}\ (\boldsymbol{\nu}\tilde{w})(S \mid s : \tilde{h})$ where S is the parallel composition of two or more processes having active prefixes, as $S_i\langle s[\hat{\mathtt{p}}_i]\rangle, i \in [1..n], n \geq 2$, where $S \equiv S_1 \mid S_2 \mid ... \mid S_i \mid s : \tilde{h}$ that do not form a redex.

Theorem 3.4.60 (Communication-safety)**.** A typeable program never reduces to an error process.

Proof. By type preservation, it suffices to show that typable programs are not errors. The proof is by reductio ad absurdum, assuming error processes are typable. Suppose that $\Gamma; C \vdash \mathsf{def}\ D\ \mathsf{in}\ (\boldsymbol{\nu}\tilde{w})(R \mid s : \tilde{h}) \triangleright \Delta$. Analysing the derivation tree for the process, we conclude that $\Gamma; C \vdash R \mid s : \tilde{h} \triangleright \Delta$.

When R is the parallel composition of two or more processes having active prefixes $R \equiv R_1 \mid R_2 \mid ... \mid R_i \mid s : \tilde{h}$ that do not form a redex. We concentrate on the case of two processes; the remaining cases are the same. Let $R \equiv R_1 \mid R_2 \mid s : \tilde{h}$ then the cases are the following:

Case: Reception(s)-Select Queue By rule [TRConc], we have that $\Gamma; C \vdash R_1 \triangleright \Delta_1$, $\Gamma \vdash R_2 \triangleright \Delta_2$ and $\Gamma; C \vdash s : \tilde{h} \triangleright \Delta_3$ where $\Delta = \Delta_1, \Delta_2 \star \Delta_3$. By rule [TRcv] and [TRSel], we have that $\Delta_1', s[\hat{\mathtt{p}}] :?\langle \hat{\mathtt{q}}, U\rangle; T = \Delta_1$, $\Delta_2', s[\hat{\mathtt{q}}] :?\langle \hat{\mathtt{p}}, U\rangle; T' = \Delta_2$ and $s[\hat{\mathtt{q}}] : \oplus\langle \hat{\mathtt{p}}, l\rangle; \Delta_3' = \Delta_3$ (Symmetric is the case when $s[\hat{\mathtt{p}}] : \oplus\langle \hat{\mathtt{q}}, l\rangle; \Delta_3' = \Delta_3$); thus, $\Delta = \Delta_1', \Delta_2', s[\hat{\mathtt{p}}] :?\langle \hat{\mathtt{q}}, U\rangle; T, s[\hat{\mathtt{q}}] :?\langle \hat{\mathtt{p}}, U\rangle; T'; s[\hat{\mathtt{q}}] : \oplus\langle \hat{\mathtt{p}}, l\rangle; \Delta_3' = \Delta_1', \Delta_2', s[\hat{\mathtt{p}}] :?\langle \hat{\mathtt{q}}, U\rangle; T, s[\hat{\mathtt{q}}] : \oplus\langle \hat{\mathtt{p}}, l :?\langle \hat{\mathtt{p}}, U\rangle; T\rangle'; \Delta_3'$. No causality can be formed between the two prefixes $s[\hat{\mathtt{p}}] :?, s[\hat{\mathtt{q}}] : \oplus$, resulting in non well-formed global type. The other case defines the two prefixes (with casual edges IO, IO) in two co-occurring causalities that cannot be ordered (Note that at session initiation time, the global type is not parameterised any more; it has a similar definition as in CSMS with the only difference that does not contain channels in the definition of causalities). Hence, we have a contradiction that $\Gamma; C \vdash \mathsf{def}\ D\ \mathsf{in}\ (\boldsymbol{\nu}\tilde{w})(R \mid s : \tilde{h}) \triangleright \Delta$ is typeable. Note that by assumption the queue s contains only labels, so the queues generated by the permutation of the messages are included in this case.

Case: Branching(s)-Send Queue By rule [TRConc], we have that $\Gamma; C \vdash R_1 \triangleright \Delta_1$, $\Gamma \vdash R_2 \triangleright \Delta_2$ and $\Gamma; C \vdash s : \tilde{h} \triangleright \Delta_3$ where $\Delta = \Delta_1, \Delta_2 \star \Delta_3$. By rule [TBr] and [TRSend], we have that $\Delta_1', s[\hat{\mathtt{p}}] : \&\langle \hat{\mathtt{q}}, \{l_i : T_i\}_{i\in I}\rangle = \Delta_1$, $\Delta_2', s[\hat{\mathtt{q}}] : \&\langle \hat{\mathtt{p}}, \{l_i : T_i'\}_{i\in J}\rangle = \Delta_2$ and $s[\hat{\mathtt{q}}] :!\langle \hat{\mathtt{p}}, U\rangle; \Delta_3' = \Delta_3$ (Symmetric is the case when $s[\hat{\mathtt{p}}] :!\langle \hat{\mathtt{q}}, U\rangle; \Delta_3' = \Delta_3$); thus, $\Delta = \Delta_1', \Delta_2', s[\hat{\mathtt{p}}] : \&\langle \hat{\mathtt{q}}, \{l_i : T_i\}_{i\in I}\rangle, s[\hat{\mathtt{q}}] : \&\langle \hat{\mathtt{p}}, \{l_i : T_i'\}_{i\in J}\rangle; s[\hat{\mathtt{q}}] :!\langle \hat{\mathtt{p}}, U\rangle; \Delta_3' = \Delta_1', \Delta_2', s[\hat{\mathtt{p}}] : \&\langle \hat{\mathtt{q}}, \{l_i : T_i\}_{i\in I}\rangle, s[\hat{\mathtt{q}}] :!\langle \hat{\mathtt{p}}, U\rangle; \&\langle \hat{\mathtt{p}}, \{l_i : T_i'\}_{i\in J}\rangle; \Delta_3'$. No causality can be formed between the two prefixes $s[\hat{\mathtt{p}}] : \&, s[\hat{\mathtt{q}}] :!$, resulting in non well-formed global type. The other case

defines the two prefixes (with casual edges IO, IO) in two co-occurring causalities that cannot be ordered. Hence, we have a contradiction that $\Gamma; C \vdash \mathsf{def}\ D\ \mathsf{in}\ (\boldsymbol{\nu}\tilde{w})(R \mid s : \tilde{h}) \triangleright \Delta$ is typeable. Note that by assumption the queue s contains only values, so the queues generated by the permutation of the actions are included in this case.

Case: Reception-Send queue By rule [TRConc], we have that $\Gamma; C \vdash R_1 \triangleright \Delta_1$, $\Gamma \vdash R_2 \triangleright \Delta_2$ and $\Gamma; C \vdash s : \tilde{h} \triangleright \Delta_3$ where $\Delta = \Delta_1, \Delta_2 \star \Delta_3$. By rule [TRcv] and [TRSel], we have that $\Delta_1', s[\hat{\mathtt{p}}] :?\langle \hat{\mathtt{q}}, U\rangle; T = \Delta_1$, $\Delta_2', s[\hat{\mathtt{q}}] :?\langle \hat{\mathtt{p}}, U\rangle; T' = \Delta_2$ and $s[\hat{\mathtt{q}}] :!\langle \hat{\mathtt{p}}, U'\rangle; \Delta_3' = \Delta_3$; thus, $\Delta = \Delta_1', \Delta_2', s[\hat{\mathtt{p}}] :?\langle \hat{\mathtt{q}}, U\rangle; T, s[\hat{\mathtt{q}}] :?\langle \hat{\mathtt{p}}, U\rangle; T'; s[\hat{\mathtt{q}}] :!\langle \hat{\mathtt{p}}, U'\rangle; \Delta_3' = \Delta_1', \Delta_2', s[\hat{\mathtt{p}}] : ?\langle \hat{\mathtt{q}}, U\rangle; T, s[\hat{\mathtt{q}}] :!\langle \hat{\mathtt{p}}, U'\rangle; ?\langle \hat{\mathtt{p}}, U\rangle; T; \Delta_3'$. No causality can be formed between the two prefixes $s[\hat{\mathtt{p}}] :?, s[\hat{\mathtt{q}}] :!$ as $U \neq U'$, resulting in non well-formed global type. The other case defines the two prefixes (with casual edges IO, IO) in two co-occurring causalities that cannot be ordered. Hence, we have a contradiction that $\Gamma; C \vdash \mathsf{def}\ D\ \mathsf{in}\ (\boldsymbol{\nu}\tilde{w})(R \mid s : \tilde{h}) \triangleright \Delta$ is typeable. Note that by assumption the queue s contains only labels and values that cannot form a causality with the reception action, so the queues generated by the permutation of the messages are included in these case.

Case: Branching-Select Queue By rule [TRConc], we have that $\Gamma; C \vdash R_1 \triangleright \Delta_1$, $\Gamma \vdash R_2 \triangleright \Delta_2$ and $\Gamma; C \vdash s : \tilde{h} \triangleright \Delta_3$ where $\Delta = \Delta_1, \Delta_2 \star \Delta_3$. By rule [TBr] and [TRSend], we have that $\Delta_1', s[\hat{\mathtt{p}}] : \&\langle \hat{\mathtt{q}}, \{l_i : T_i\}_{i\in I}\rangle = \Delta_1$, $\Delta_2', s[\hat{\mathtt{q}}] : \&\langle \hat{\mathtt{p}}, \{l_i : T_i'\}_{i\in J}\rangle = \Delta_2$ and $s[\hat{\mathtt{q}}] : \oplus\langle \hat{\mathtt{p}}, l\rangle; \Delta_3' = \Delta_3$; thus, $\Delta = \Delta_1', \Delta_2', s[\hat{\mathtt{p}}] : \&\langle \hat{\mathtt{q}}, \{l_i : T_i\}_{i\in I}\rangle, s[\hat{\mathtt{q}}] : \&\langle \hat{\mathtt{p}}, \{l_i : T_i'\}_{i\in J}\rangle; s[\hat{\mathtt{q}}] : \oplus\langle \hat{\mathtt{p}}, l\rangle; \Delta_3' = \Delta_1', \Delta_2', s[\hat{\mathtt{p}}] : \&\langle \hat{\mathtt{q}}, \{l_i : T_i\}_{i\in I}\rangle, s[\hat{\mathtt{q}}] : \oplus\langle \hat{\mathtt{p}}, l : \&\langle \hat{\mathtt{p}}, \{l_i : T_i'\}_{i\in J}\rangle; \Delta_3'$. No causality can be formed between the two prefixes $s[\hat{\mathtt{p}}] : \&, s[\hat{\mathtt{q}}] : \oplus$ as $l \notin l_i, i \in I$, resulting in non well-formed global type. The other case defines the two prefixes in two co-occurring causalities that cannot be ordered. Hence, we have a contradiction that $\Gamma; C \vdash \mathsf{def}\ D\ \mathsf{in}\ (\boldsymbol{\nu}\tilde{w})(R \mid s : \tilde{h}) \triangleright \Delta$ is typeable. Note that by assumption the queue s contains only values and labels that are not part of the branching prefix, so the queues generated by the permutation of the actions are included in these case.

For the other cases of more than two processes, the structure of the proof is the same with the difference that the reception, branching constructs are repeated more than twice in respectively each of the forementioned cases. We note that $i-1$ prefixes of reception or branching can not form a causality with respectively the select or send prefix, resulting in a non well-formed global type. For the case which the two prefixes (with casual edges IO, IO) are defined in two co-occurring causalities then they cannot be ordered. This is impossible as G is well-formed. Hence, we have a contradiction that $\mathsf{def}\ D\ \mathsf{in}\ (\boldsymbol{\nu}\tilde{w})(R \mid s : \tilde{h})$ is typeable. □

Below, we provide as a conjecture the property of progress: a general expression has the progress property if in all configurations where it is provided a suitable context either (1) it does not contain session channels or (2) it can be further reduced.

Conjecture 3.4.61 (Progress)**.** A general expression E has the progress property if $E \rightarrow^{\star} E'$ implies either E' does not have session channel or $E'|E'' \rightarrow$ for some E'' such that $E'|E''$ is well-typed and $E'' \not\twoheadrightarrow$.

Although, we do not have a formal proof, we believe the system satisfies the standard progress property. Indeed, our system benefits from the proof of progress for well-typed processes willing to start a session by Bettini et al. (2008). To complete the proof, we need to ascertain that a well-typed program reduces to the above processes and that a well-typed iterative behaviour reduces further. We leave the proof of progress for future work.

3.5 Another System

The idea of parameterised session types has been modeled in another system by Yoshida et al. (2010)[2]. That model is based after that of Bettini et al. (2008) as CPS but extends it in an amorphous way; that is, the calculus in that system does not provide concepts that match those of mainstream languages and so, does not provide a framework incorporating parameterised session types into those languages. The key design choices of the type system present the shortfalls that limit its utility in practice. Firstly, the type system requires *programmers to write the process types*, in addition to the global type, for type-checking. Secondly, the coherence of the process types with respect to the global type is ensured by an *equivalence relation* for every value of each parameter present in the global type. The former aspect of the type system increases the programming effort and the latter diverges from the efficient typing strategy of the fundamental work of global session types by Honda et al. (2008b), where the global type is projected onto participants to obtain the types for typechecking of processes. Thirdly, the type system restricts the computing power of programs, allowing values of parameters to range over finite sets of natural numbers, e.g. parameter n:$\{m$:$\mathsf{nat} \mid 0 \leq m \leq 1000\}$ is typed by a bounded set of natural numbers. Finite sets are necessary to provide decidability for the type system, as termination of

[2]The author of this dissertation is a co-author of that paper, contributing with the idea and name of parameterised session types and formalisation of parameters in the syntax of global types and processes.

the equivalence algorithm depends on those sets' cardinality. The upper-bound of the set reflects the maximum capacity of the hardware resources to program date, not matching the purpose of innovative dynamic computing platforms such as clouds where hardware resources flex in response to demand.

For example, the program of the Ring pattern in that system is defined as:

Global type: $\Pi n : I.(\mathbf{R}\ \mathtt{W}[0] \rightarrow \mathtt{W}[n] : \langle \mathsf{nat} \rangle.\mathsf{end}\ \lambda i.\lambda \mathbf{x}.\mathtt{W}[i+1] \rightarrow \mathtt{W}[i] : \langle \mathsf{nat} \rangle.\mathbf{x}\ n)$

End-point type:
$\mathsf{if}\ \mathtt{p} = \mathtt{W}[n]\ \mathsf{then}\ !\langle \mathtt{W}[n-1], \mathsf{nat} \rangle; ?\langle \mathtt{W}[0], \mathsf{nat} \rangle; \mathsf{else}$
$\mathsf{if}\ \mathtt{p} = \mathtt{W}[0]\ \mathsf{then}\ ?\langle \mathtt{W}[1], \mathsf{nat} \rangle; !\langle \mathtt{W}[n], \mathsf{nat} \rangle; \mathsf{else}$
$\mathsf{if}\ \mathtt{p} = \mathtt{W}[i]\ \mathsf{then}\ ?\langle \mathtt{W}[i+1], \mathsf{nat} \rangle; !\langle \mathtt{W}[i-1], \mathsf{nat} \rangle;$

Process:
$\Pi n.(\mathbf{R}(\bar{a}[\mathtt{W}[n], .., \mathtt{W}[0]](y).y?(\mathtt{W}[n-1], z); y!\langle \mathtt{W}[0], z \rangle; \mathsf{end}$
$\quad \mid a[\mathtt{W}[0]](y).y!\langle \mathtt{W}[1], v \rangle; y?(\mathtt{W}[n], z); \mathsf{end})$
$\quad \lambda i.\lambda X.(a[\mathtt{W}[i+1]](y).y?(\mathtt{W}[i], z); y!\langle \mathtt{W}[i+2], z \rangle; \mathsf{end} \mid X) \quad n)$

where a special binder Π is introduced in the global type syntax, prohibiting syntactically the relation between the number of participants in global types and processes. Both the global type and end-point types are part of the user-defined program. For every value of the finite set I, instances of global types and end-point types are generated to check conformance of the latter with the former. The index expressions of that work are more general than in CPS, including a more sophisticated mathematical definition and more than one index variable per index expression. This expressivity comes at the cost of having values of parameters to range over finite sets of natural numbers in the conservative type system. In the first process, the list of principals is not mathematically defined, weakening the properties of the type system; recalling from the type system in the previous section, the mathematical definition of principals is used to well-type the process with a session initiation prefix so that the number of principals in the global type is the same as the one in processes. Also, the kinding system allows kinded subterms in the primitive recursive global types, breaking the soundness of the system. We illustrate the problem with a global type kinded by Type and which after two reductions is not kinded:

$\mathbf{R}\ \mathbf{R}\ \mathsf{end}\ \lambda i.\lambda \mathbf{x}.\mathtt{W}[i] \rightarrow \mathtt{W}[i+1] : \langle U \rangle.\mathbf{x}$
$\quad j.\lambda y.\mathbf{R}\ \mathsf{end}\ \lambda k.\lambda z.\mathtt{W}[j] \rightarrow \mathtt{W}[j+1] : \{true : y, false : z\}\ 1\ 1 \longrightarrow$
$\mathbf{R}\ \mathsf{end}\ \lambda k.\lambda z.\mathtt{W}[1] \rightarrow \mathtt{W}[2] : \{true : \mathbf{R}\ \mathsf{end}\ \lambda i.\lambda \mathbf{x}.\mathtt{W}[i] \rightarrow \mathtt{W}[i+1] : \langle U \rangle.\mathbf{x}, false : z\}\ 1 \longrightarrow$
$\mathtt{W}[1] \rightarrow \mathtt{W}[2] : \{true : \mathbf{R}\ \mathsf{end}\ \lambda i.\lambda \mathbf{x}.\mathtt{W}[i] \rightarrow \mathtt{W}[i+1] : \langle U \rangle.\mathbf{x}, false : \mathsf{end}\}$

Despite the practical shortfalls, the system presents a different approach to modelling parameterised session types, possessing the progress property and considering an interesting

example: the Fast Fourier Transformation over a 3D-Mesh.

CPS does not suffer from these shortfalls as it provides an idiom of role and a different type system. Roles provide a similar concept to classes in class-based languages and so, offer a way of incorporating parameterised session types into a mainstream language such as Java and C#. CPS's static type system follows the efficient typing strategy and programming methodology of multiparty session types: programmers first define the global type of the intended pattern and then define each role of it; roles are validated by type-checking through (1) projection of the global type onto the principals and (2) sorting and **R**-elimination of role types. We achieve strict global type annotations in programs and efficient type-checking by extending the multiparty projection algorithm to parameterised principals. Values of parameters range over infinite sets of natural numbers, providing full computation power of programs that implement parameterised communication patterns.

Chapter 4

Related Work

CSMS and CPS build on previous systems of session types. CSMS extends the study on session types by using ideas originated from structured interactions, including "choreography", while CPS introduces the idea of parameterised session types as an evolution of the ideas on multiparty session types. Both the models present type systems to guarantee communication-safety in mobile processes.

Section 4.1 reviews a wide-ranging collection of related work on session types, including: binary, multiparty, parameterised session types. Section 4.2 describes and compares the metaphors of choreography and orchestration; metaphors that explain how to specify structured interactions in processes. Other type systems that guarantee communication-safety in mobile processes are examined in Section 4.3. Finally, Section 4.4 surveys other well-known communication systems and their type systems, including systems designed over shared memory.

4.1 Session Types

4.1.1 Binary Session Types

Binary session types by Takeuchi et al. (1994) and Honda et al. (1998) are the fore-runners of multiparty session types. They capture the communication structure between two processes and, as global types, they are part of the program annotations. Processes are validated by type-checking and reciprocity of session types. That is, the session types of each of the two processes must be reciprocal. The first work is modeled on a variant of

the π-calculus for structured communications over synchronous communications, while the second one extends the first by introducing higher-order communication over asynchronous communications. Channels are considered linear in the sense of Girard (1987) linear logic —shared only by the two complementary processes— to avoid aliasing.

Research on binary session types has been ongoing for over a decade, adding additional features such as subtyping by Gay and Hole (2005), a practical form of higher-order communication by Yoshida and Vasconcelos (2007), higher-order processes by Mostrous and Yoshida (2007), bounded polymorphism by Gay (2008) and multithreading for functional languages by Vasconcelos et al. (2004, 2006). The following paragraphs describe other works in the field, divided by area of study, where each paragraph presents the works in chronological order.

Binary session types are studied for an object calculus by Dezani-Ciancaglini et al. (2006) and Coppo et al. (2007), including higher-order communication and, session linearity in the former and progress in the latter. A more significant progress property inside a single session is proposed in the system of Dezani-Ciancaglini et al. (2007), freeing the restriction of Coppo et al. (2007) that only overlapping sections can be nested and that outer sessions can proceed only when inner ones have ended. Gay et al. (2010) provide a more general theory of object-based session-programming, including inheritance, subtyping and typestate (DeLine and Fähndrich (2004)); typestate is used to control the sequence of method calls that represent the session types of an object's session channels. The theory of binary session types is examined for ambients by Garralda et al. (2006), proving a basic theory of session fidelity.

A general algorithm to infer binary session types from processes has been proposed by Mezzina (2008), including types for higher-order communication; subsequently, the types returned by the algorithm are checked for reciprocity to guarantee deadlock-freedom within a session. Inference of session types is an appealing typing process for binary session types due to their non-intuitive syntax and simplicity. The syntax of binary session types is based on bangs and question marks similar as the other systems mentioned in the introduction; for example, the binary session types of the ATM with the Debit-Card and with the Bank in the ATM–Debit-Card–Bank example are, respectively, $1!\langle int\rangle; 1\&\{ok : \mathsf{end}, ko : \mathsf{end}\}$ and $2 \oplus \{withdraw : 2!\langle int\rangle; \mathsf{end}, balance : 2?(int); \mathsf{end}, deposit : 2!\langle int\rangle; \mathsf{end}\}$.

SJ (Hu et al. (2008)) was described in Section 2.4.2. In addition to the features presented in Section 3.2.2, SJ supports (1) multicast session-iteration to control a number

of processes simultaneously, (2) programs to make use of the best transport available without modifying the programs themselves (Hu et al. (2009)), (3) high-level message types (Bejleri et al. (2009)) to improve readability and type-safety, and (4) subtyping to provide reciprocity between select-branch constructs. The latter is, a session type $\oplus\{l_1 : T_1, ..., l_n : T_n\}$ is reciprocal to $\&\{l_1 : U_1, ..., l_n : U_n, ..., l_{n+m} : U_{n+m}\}$ if T_i is reciprocal to U_i for $1 \leq i \leq n$. The implementation uses the Polyglot class library of Nystrom et al. (2003) —a front end Java compiler— to build constructs of SJ as extensions of Java without copying the framework code. Firstly, the SJ compiler statically verifies correctness by checking each session implementation against its declared type. Secondly, at session initiation time, the runtime checks on each peer to ascertain that the session type of the other end-point is reciprocal to its (peer) session type. If successful, the session is established, otherwise both parties raise an exception (`SJIncompatibleSessionException`) and the session is aborted. During the execution of a session, if an exception is raised, due to communication failure at one or both sides of an enclosing session-try scope, a protocol is responsible for propagating a failure signal to all other active sessions within the same scope thus maintaining consistency across such dependent sessions. Despite this strong work, SJ is not expressive enough to represent sessions of an arbitrary number of workers as shown in the Monte Carlo algorithm. The lack of parameters in binary session types results in inefficient implementation of parallel algorithms as in n-Body (Bejleri et al. (2009)) where the session between the first and last worker in the pipeline opened and closed in every iteration of the algorithm. Also, the use of threads in the Dense Linear Algebra algorithms to simulate an arbitrary number of Workers resulted in the slow down of the algorithm due to the creation of threads.

Bejleri et al. (2009) have demonstrated the expressiveness, productivity and performance benefits of session-based programming in SJ through the implementation of parallel algorithms, most of them discussed in Chapter 3. SJ programs are guaranteed to be free from type and communication errors by the semantics of session communication and static session type checking. The first two points of common MPI errors (described in the introduction), (1) invalid actions before `MPI_Init` and after `MPI_Finalize` and (2) unmatched `MPI_SEND` and `MPI_RECV`, are directly prevented by the properties of session types. The third point related to concurrency does not occur in SJ as the compiler disallows sharing of session socket objects (implicitly `noalias`), and safely controls message copying/linear transfer via `noalias` types. They have also observed that SJ programs perform competitively against other Java communication runtimes. The authors evaluated that SJ programs perform better than lower-level, non communication-safe message-

passing systems such as MPJ Express. Hu et al. (2008) established that in certain cases, SJ programs can out-perform their counterparts implemented in communication-safe systems such as RMI (Grosso (2001))— remote procedure call in object oriented.

Neubauer and Thiemann (2004) have implemented session types in Haskell. Session types are encoded in type constructs of the language, avoiding intervention in the language abstract tree. The type preservation property is proved to hold in the formal model of the implementation. Pucella and Tov (2008) propose a portable, stack-based implementation of session types in Haskell using indexed monads. Binary session types have been implemented in Haskell by Sackman and Eisenbach (2008) using monads, resulting in a pure implementation without complicating the language as in the fore-mentioned works; however, that work provides a larger set of features such as higher-order communications.

4.1.2 Multiparty Session Types

Multiparty session types were introduced by Honda et al. (2008b) as mentioned in the introduction. The calculus extends the one of binary session types by modifying the primitives for session initiation to multiparty sessions and adding a message queue. The new primitives reflect the multiparty nature of handshake at session initiation and the message queue is used by the operational semantics to provide a media of communication over a group of participants, maintaining the order of messages sent asynchronously within the group of processes. In contrast to binary session types, generic types (Kobayashi et al. (2000)), behaviour types (Amtoft et al. (1997)) and contracts (Castagna and Padovani (2009)), global session types of the initial work and CSMS describe the order of messages between several participants that is part of the program logic as explained in Section 1.2.

Despite the strong work on binary session types, multiparty session types offer an intuitive, structural type annotation to write communication-safe concurrent, distributed programs. This is illustrated through the ATM–Debit-Card–Bank example, given the global type and the binary session types of the ATM process (both sessions) in the introduction. Another advantage of multiparty session types is that the linearity property is more expressive since it allows channels to be shared safely across different processes, while in binary session types, session channels are shared only between two complementary processes.

CSMS is modelled on the system of Honda et al. (2008b), extending the latter with a simpler calculus, multicasting of values and labels, a practical form of higher-order

communication, an intuitive, elegant definition of linearity and other features discussed throughout Chapter 2. Bettini et al. (2008) present a simpler version of the multiparty session type theory in a setting where channels are not shared between different communications, removing channels from the syntax of global types. The difference with the type and process constructs of Bettini *et al.* and Honda *et al.* were discussed in Section 3.4.

Mostrous et al. (2009) enrich the typing discipline of multiparty session types for optimisation purposes. Through a subtyping relation on the end-point types returned by projection, the type system allows programs that specify a different order of "send"s from the one in the global type for the sake of performance; e.g., a packet-size message can be sent before small ones for greater speed-up. We plan to include this feature in CPS as we shall see later in the future work section.

Multiparty session types are used to type service-oriented sessions by Bruni et al. (2008). That calculus proposes communication inside and outside a session, and locations to model merging of two running sessions. Type preservation and progress properties are not provided for the formal model, leaving them as future work.

Recalling from the introduction, global types not only offer a typing discipline but also a language to describe the architecture of a system, related to communications, through an intuitive syntax. Thus, a type inference algorithm is not of interest in multiparty and parameterised sessions. Also, inference of global types is more challenging than binary session types as the information to be deduced from processes consists of the interaction structure, including order, between several and an arbitrary number of peers. Instead, an appealing tool for multiparty session types is code generation.

Code generation from global description is defined in a theory of end-point projection (EPP) of Carbone et al. (2007a,b). The global calculus given in EPP is used to describe interactions of a session globally and subsequently project it onto the participants to obtain the running processes. In addition to the constructs of CSMS global types, the global calculus offers assignment of local variables in processes and independent choice over global behaviours. The additional constructs and syntactic sugar present in the syntax are useful when building web services protocols. However, the global calculus does not support higher-order communication and multicasting as in CSMS. A number of principles check the behaviours described by the global calculus to ensure that the behaviour of the processes, returned by projection, will be the same as the one in the global calculus. Those checks result in a conservative language that does not allow programmers to describe interesting

behaviours. Also, channels are not shared between different communications and so no linearity property is needed. However, this further restricts the use of multithreading in the global calculus.

4.1.3 Parameterised Session Types

As mentioned in the introduction, the multiparty session type theory (for both the synchronous and asynchronous systems) excels at addressing sessions of a fixed number of participants but fails to address sessions of an arbitrary number of participants. CPS and its type system evolve from the theory of multiparty sessions, addressing sessions of an arbitrary number of participants and preserving its programming methodology and typing strategy. None of the systems mentioned in the previous subsections is expressive enough to model communication patterns as global types and so, benefit from the typing strategy of parameterised session types. The idea of parameterised session types has been modelled recently in the work of Yoshida et al. (2010), which was compared with CPS in Section 3.5.

CPS is modelled on the system of Bettini et al. (2008), extending with (1) parameters the syntax of participants in processes and global types, and (2) recursive operator the syntax of global types and processes; processes are defined as second-class constructs, representing general expressions as first-class, to support the idiom of role as described throughout Section 3.4. The **R** operator used in both CPS and the fore-mentioned system was introduced by Gödel in System T (Alves et al. (2010)). The idea of using the **R** operator in global types and roles comes from Nelson (1991)'s work on adding primitive recursion to the lambda calculus which, as he states, is a finite representation of the $\mathcal{T}^{\infty}$ system of Tait (1965) and Martin-Lőf (1972). Our use of the **R** operator models and types parameterised sessions.

Indexes in types are used to type arrays, lists and stacks in a dependency type system —types indexed by terms— for ML (DML) to ensure correct use of "remove", "delete" methods in the fore-mentioned data structures. DML of Xi and Pfenning (1999) uses constraints-solving of Fourier methods (Pugh and Wonnacott (1992)) to solve index equations. In CPS, we use indexes in global types to ensure correct use of indexes in the roles of a communication pattern. We have defined a set of rules to solve the index expressions based on a basic arithmetic of addition, subtraction, multiplication and a restricted form of exponentiation as defined in Section 3.4.1.

Giachino et al. (2009) present an interesting language to describe sessions of an arbitrary number of participants through two constructs: joining and leaving of participants. However, their preliminary type theory is based on a similar definition of end-point types and so, does not offer an intuitive, light-weight, structural type annotation as CPS.

4.2 Choreography and Orchestration

Choreography WS-CDL by Web Services Choreography Working Group (2005) is the first language that builds on the metaphor of *choreography* to describe interactions between participants of a session. The metaphor of choreography, introduced by Ross-Talbot (Sparkes (2006)):

> "Dancers dance following a global scenario without a single point of control"

suggests that the processes communicate following a global description and no process controls the execution of the global description. Both CSMS and CPS use this metaphor to describe the interactions between the participants of a session. WS-CDL is an XML-based language to describe the order of communications between the participants of a session as a global scenario. The aim of WS-CDL is to describe protocols independently from the programming language used to implement the participants and the platform (OS) they run. This is achieved by defining WS-CDL as a human interpretable language, like UML (Fowler and Scott (1997)) and Message Sequence Charts (MSCWG (1996)) are used to define the architecture of a system implemented in different languages. In contrast to WS-CDL, CSMS and CPS use global types not only as a blue-print of a system's architecture but also as a type system to guarantee communication-safety of the system.

"Choreography" is used to describe cryptographic protocols by Corin et al. (2007), which protect session execution from both external attackers and malicious participants. That work defines a model to program cryptographic systems rather than a typing discipline for programming languages.

Orchestration WS-BPEL of Web Services Business Process Working Group (2007) is another language used to specify structured interactions of business processes. Even though the aims of WS-BPEL are the same as the ones of WS-CDL, the metaphor of protocol description is different. WS-BPEL is designed over an orchestration metaphor

where a special process (conductor), called the Abstract Process, describes the interaction structure of each common process (musician). Consequently, each executable process implements the descriptions present in the Abstract process. The interaction structure in this metaphor is centralised, complicating the behaviour of the Abstract Process and so, going against the nature of distributed applications where each component on its own structures and shares its behaviour.

COWS by Lapadula et al. (2007) proposes a process calculus and an implementation for Web Services, using message correlation to program stateful sessions as in WS-BPEL. However, the calculus lacks the constructs to build structured interactions, making the modeling of various interaction scenarios difficult.

4.3 Guaranteeing Communication-Safety in Mobile Processes

Contracts Contracts by Castagna et al. (2009); Castagna and Padovani (2009) are another typing model of mobile processes, defined over processes and not over channels as CSMS and CPS. Consequently, they can well-type more correct programs than CSMS and CPS. However, the expressiveness of the type system comes at a practical cost. Contracts have no intuitive, structural, light-weight syntax as global types and no construct, including iterative as in CPS, to support structured communications. Thus, they do not provide a practical model, including expressivity of communication patterns, to incorporate their type theory into programming languages that support communication; e.g. the key exchange protocol, discussed in Section 3.3.3, has been augmented with additional interactions to check the end of a send-iteration and an extra private channel to send the key.

Conversation Calculus and Types The conversation calculus (CC) of Vieira et al. (2008) is based on boxed ambients (Bugliesi et al. (2004)) and not on the π-calculus as CSMS and CPS. Each process in CC is defined over a conversation context, containing the process behaviour. Conversation starts by one peer (client) calling a service located at another peer (server), where the context of the client is modeled by that of the server, and continues over the channels defined inside the conversation contexts. The conversation medium that is provided by the conversation context is similar to the shared names in CSMS and CPS. However, that medium prefixes all the terms in the reduction steps of

the conversation, modeling also the role of our session channels. The advantage that follows immediately from this design choice is the simplicity of the calculus, including syntax and operational semantics. CC does not support branching over labels as CSMS and CPS, offering a simpler calculus, and consequently the properties of the type system (discussed below) are easier to prove. Despite the simplicity of the calculus, I believe that branching over labels is the natural way to model a set of conversation paths, and is a base construct in the calculi typed in the spirit of structured interactions. CC supports joining and leaving of participants within a session. This is modeled easily in the system as both shared names and session channels are represented through the conversation context, reducing the actions of synchronizing with other members of the session and obtaining the session channels in one single action of obtaining the session context. However, in contrast to CPS, CC lacks from the definition of parameters in the syntax of channels, making it impossible to model communication patterns of an arbitrary number of participants.

The conversation types of Caires and Vieira (2010) are based on a notion of global type and end-point types in similar way as Yoshida et al. (2010). That is, programmers have to define both the global type and end-point types of a conversation, and the coherence of the latter to the former is provided through a merging relation. As mentioned in Section 3.5, this design increases the effort of programming since programmers have to define both global types and end-point types; we cannot provide any claim regarding the efficiency of the merging relation compared to our type-checking strategy since no evidence is given in that paper. The notion of global types in CC captures only the channels and types of a conversation, abstracting the identities of participants, providing a more powerful, elegant progress property (as contracts) than CSMS and CPS. However, this expressiveness comes at a practical cost: no intuitive and light-weight type syntax. CC type system supports multi-threading through the notion of parent and current conversation; i.e., a process can communicate with a process inside its context (e.g. a database) and with another one outside its context (bound by the same medium), providing a more expressive type system than the one of CSMS and CPS. CC provides an elegant typing of joining through global types that include the definition of end-point types. This is a design choice worth investigating in both CSMS and CPS: global types that include the behaviour of particular participants not from a global point of view. However, CC's type system does not provide any support to type sessions of an arbitrary number of participants. The discussion on the expressiveness of parameterised session types and types capturing joining (conversation types or session types as in Deniélou and Yoshida (2011)) is not straightforward and requires a detailed investigation that goes beyond the aim of this work; thus, relegating it

as future work.

Multipoints Multiparty sessions have been studied also by Bonelli and Compagnoni (2007). Their type system is defined over binary session types, obtained by projecting process types (CSMS end-point types) onto principals. Thus, the system provides the same shortfalls as in the previous systems, limiting its use in practice.

Generic Types Igarashi and Kobayashi (2004), and Kobayashi et al. (2000); Kobayashi (2006) present a type-system for a version of the π-calculus, extended with Booleans, conditional and pair, that ensures data-races and deadlock-freedom; the latter work gives also a type-inference algorithm in the presence of recursion. The type system types channels, similarly to contracts, keeping the frequency and order of use in the input-output actions of a process, while in CSMS and CPS, the type system types sessions, ensuring that the interaction structure at runtime follows the one of the global type. A stronger progress property holds for generic type systems (i.e. more correct programs are well-typed in Kobayashi (2006) than in session types as illustrated in Castagna and Padovani (2009) for contracts). While, CSMS and CPS type system ensure the order of communications in processes as in global types, a key element in program logic as discussed in the introduction and illustrated in Section 4.1.2. The expressiveness of generic types comes from an elegant linear analysis of channel usage over the entire definition of processes rather than focussing on particular sessions of it. However, this expressiveness comes at a practical cost that is similar to the other systems discussed in this section. Kobayashi (2006) gives the definition and typing of a role that broadcasts **true**, parameterised by a list of channels, as:

$$broadcast?(list); \mathbf{if}\ null(list)\ \mathbf{then\ 0\ else}\ head(list)!\langle \mathbf{true} \rangle \mid broadcast!\langle tail(list) \rangle$$

where $head(list)$ and $tail(list)$ return respectively the first element (head) and the remaining elements without the head of the list $list$. This example is encoded in CPS as:

$$\mathbf{R\ 0}\ \lambda i.\lambda X.head(list)!\langle \mathbf{true} \rangle \mid X\ length(list)$$

where $length(list)$ returns the length of the list $list$. The generic type system does not provide a mathematical analysis that checks the constraints between the various mathematical expressions on channels, not solving the index calculation problem discussed in Section 3.2.1. The problem becomes sharper when considering more complex communication

patterns than broadcasting. Also, generic types cannot benefit from the mathematical analyser of CPS, since they do not build on the choreography metaphor, leaving the index calculation problem unsolved.

Behaviours Behaviours by Nielson and Nielson (1994), and Amtoft et al. (1997) describe the communication behaviour that captures the causal constraints of a concurrent program through terms of a process algebra, similarly to contracts. An implementation of their system to derive behaviours is provided by Nielson et al. (1998) for CML programs, where their notion of communication pattern, expressed using behaviours, is similar to our notion of role type; CML by Reppy (1991) is a high-performance language for concurrent programming in ML that incorporates higher-order event. Behaviours have a similar typing discipline as contracts, thus they lack the same features of CSMS and CPS as contracts do.

Graph Types Yoshida (1996) proposes a typing system for the monadic π-calculus based on graphs, proving interesting properties such as full abstraction. Graph-based types express the order (dependencies) of communications and guarantee deadlock-freedom and liveness of channels. The system does not provide constructs to build and capture structured interactions as in CSMS and CPS, however it represents an interesting type annotation that is worth investigating in our framework.

4.4 Other Communication Systems

MPI MPI (Gropp et al. (1999)) is a message-passing API to program parallel computers with a rich set of communication primitives, suffering from a low-level API, common errors such as index calculation, invalid actions before initiation and after ending a session, and unmatched "send" and "receive". Since MPI has an extensive library of functions developed over 15 years, many of its constructs are not yet directly supported in CSMS or CPS. However, many of these features can be encoded into session constructs, as shown by Bejleri et al. (2009). MPI is designed as a portable API specification to be implemented for varying host languages. The design of verification techniques for a host language is difficult given the low-level nature of many MPI functions. The common MPI errors discussed in the introduction are prevented by session programming and typing:

- **Doing things before** `MPI_Init` **and after** `MPI_Finalize`. In session programming, the static type system does not allow an action before the session has been initialised or closed.
- **Unmatched** `MPI_Send` **and** `MPI_Recv`. Session programming prohibits these errors statically by the reciprocal property of session types: for each "send" on one side corresponds a "receive" on the other and vice versa.
- **Concurrency issues**. The static type system of session types does not allow multi references and sharing of session channel.

Contracts Contracts by Fähndrich et al. (2006) are used to describe and type the interfaces of the components of the Singularity OS that communicate via message-passing, designed over shared memory. Their concept of contract is similar to binary session types, where each endpoint interface specifies the methods required for each state of the contract and the messages consist of asynchronous method invocation and information on the arguments.

StreamFlex StreamFlex is a real-time language, developed by Spring et al. (2007), providing a stream Java API that guarantees sub-millisecond response times. Objects specialised for communication are classified in the heap by ownership types to ensure linearity of messages. StreamFlex proposes an interesting framework of using ownership types (Clarke et al. (1998)) to control the topology of an object graph, expressing the constraints as a pluggable type system (Bracha (2004)).

XMem XMem (Wegiel and Krintz (2008)) as StreamFlex provides another managed memory model for concurrent programming, supporting isolation and sharing of objects between several JVMs. Shared memory attach/detach and channel establishment require global synchronisation across runtimes. Managed Runtime Environments apply the same types for shared objects over shared memory to guarantee type-safety in XMem.

XMem, StreamFlex and Contracts guarantee communication safety for a particular set of objects (components) and do not provide a general typing strategy for message-passing programs as CSMS and CPS. MPI provides a message-passing API but the implementations do not provide any static analysis to detect communication errors at compile time mainly due to its low-level API.

Chapter 5

Conclusion

"*In everything... uniformity is undesirable. Leaving something incomplete makes it interesting, and gives one the feeling that there is room for growth... Even when building the imperial palace, they always leave one place unfinished.*"

Japanese Essays in Idleness
14^{th} Century[1]

We conclude by evaluating the results presented in this dissertation through the hypotheses given in the introduction and discuss possible future work.

5.1 Evaluation

The thesis of this dissertation stated in the introduction:

> *Parameterised session type theory can be used to guarantee communication-safety in sessions of an arbitrary number of participants, typically represented as communication patterns, of mobile processes, supporting (1) a concept similar to class—role, (2) an intuitive, structural, light-weight type annotation of programs and (3) an efficient, liberal typing.*

[1] Citation first heard from Marcus du Sautoy.

is supported by five validated hypotheses. The order of the hypotheses in the introduction followed the presentation order of the sections validating them. In this section, the order follows the argument of parameterised session types, including its intuitive idea and formal model, and the studies that support it.

Hypothesis III Parameterised session types describe rich patterns of communication, namely Star, Ring, Tree, Mesh (Sections 3.1 and 3.2), used ubiquitously in the development of distributed applications as demonstrated in the parallel algorithms of Sections 3.3.1, 3.3.2, and the data exchange protocol of Section 3.3.3. The theory offers an expressive calculus and type system, allowing constructs similar to the *for* loop (Section 3.2.2, 3.4.6) to be safely part of the calculus, enriching the set of well-typed programs. Binary and multiparty session types cannot express communication patterns of an arbitrary number of participants, resulting in non-reusable programs: representing single computations in a single class (Sections 3.2.2), inefficient implementations: creation of threads unnecessary to the program logic (Section 3.3.2). The fore-mentioned shortfalls are present also in contracts, resulting in inefficient implementations: creation of redundant messages and conditions (Section 3.3.3). Another advantage discovered of parameterised session types is that index calculation of principals in global types is less complex than the one in roles due to the global representation of interactions, following a straight line. Our type system exploits that fact to type-check the indexes in roles, offering a method of controlling the main source of errors in MPI: index calculation (Section 3.2.1).

Hypothesis V The formal model of parameterised session types (CPS and its type system) is well-established (Section 3.4), extending the model of multiparty session types (Section 2.6) through (1) declarative type and programming constructs of parameterised participants, recursive operator (Section 3.4.1), and (2) elegant, efficient concepts, namely projection, sorting and **R**-elimination (Section 3.4.4). In CPS, programmers first define the global type of the intended pattern and then define each role of it; roles are then validated by type-checking in $O(p * d * n \log n)$ computational steps through (1) projection of the global type onto the principals, (2) sorting and R-eliminating of role types returned by (1). CPS offers a liberal type system, allowing values of parameters to range over infinite sets of natural numbers to provide full computation power of programs. Type preservation under reduction of expressions and communication-safety hold for the type system (Section 3.4.7).

Hypotheses I and II Parameterised session primitives and types build over the ones of multiparty sessions. Multiparty session types stand out from other type-systems of mobile processes by an intuitive, structural, light-weight type syntax and efficient type-checking strategy. Describing the structure of interactions from a global point of view using simply names and arrows allows us to describe a component of the program logic: order of communications as demonstrated by CSMS. Multiparty session types have a light-weight type annotation of programs since only one global type types a program independently of the number of processes defining the latter (Section 2.1). CSMS overcomes the practical shortfalls of the initial work of Honda *et al.* by defining a simple, minimal calculus (Section 2.1) well-established by the formal model (Section 2.6) which served as the foundation of CPS, introducing multicast send of values and labels, providing a practical form of higher-order communications and linearity (Sections 2.3, 2.4, 2.5). These features were not considered in CPS; higher-order communications and linearity are omitted to facilitate the session calculus for the study of parameterised session types, while multicasting becomes an unused feature in CPS. However, higher-order communications and linearity in CSMS can be easily incorporated into CPS due to their practical and intuitive definition. In CSMS, programmers first define the global type of the intended interaction and then define each process of it; processes are then validated by type-checking in $O(n^3)$ computational steps through projection of the global type onto the principals (Section 2.6.4). Preservation under reduction, communication-safety and session fidelity of processes with respect to global types both hold for the type system of CSMS (Section 2.6.5).

Hypothesis IV The idiom of roles introduced in CPS is an abstraction of end-points, describing the nature of a communication pattern and the behaviour that all runtime processes share. It matches elegantly the idiom of classes (Section 3.1) as abstraction entities over participants, offering a practical concept on how to incorporate parameterised session types into mainstream languages such as Java and C#; additional features such as communication media and booting of programs must be supported to provide the proper framework to implement parameterised session types as we shall see in the next section. However, a concrete implementation of parameterised session types on top of the fore-mentioned languages will fully evaluate the formal definition of roles.

5.2 Future Work

The future directions of this work are guided by (1) the knowledge gained from this study,

including the exploration of MPI, (2) feedback from the reviewers and attendees of PLACES'08, PLACES'09 and ICFEM'10 and (3) discussions with students and academics at TiC'08 summer school.

5.2.1 Implementing a Communication-Safe Library

The next step in developing this work is the implementation of CPS constructs and its type system as a library of a mainstream class-based language. This is a necessary, long-term effort to evaluate the theory of parameterised session types. The runtime of the library must support constructs for structuring interactions, such as branching and session channels such as communicators in MPI, e.g. MPI_COMM_WORLD. In contrast to SJ, we may not need additional iterative loops as `in-outwhile` to support iteration as our system can type *for* loops. However, we will need recursion to describe infinite behaviour. During the design of session channels, we plan to identify the design issues of multiparty handshake at session initiation; in SJ, the handshake between two processes was granted by the semantics of the TCP socket. Another challenge to the runtime is the running/instantiating of the roles (classes) that simulates the semantics of the **R** operator. One way to address this issue may be to define a script that specifies the roles with the number of instances and the machines to run the instances created. This approach is similar to MPJ Express running of programs but differs in the script format as the latter has only one class per program, resulting from the SPMD programming model.

We have found that the four modes of communication in MPI: *standard*, send and receive block on their respective buffers, *synchronous*, send and receive operations synchronise, *ready*, programmer notifies the system that a receive has been posted, and *buffered*, user manually handles send buffers, greatly increase productivity in programming parallel algorithms. This is a necessary feature to provide for a competitive, safe library of communications.

The compiler of the library will project the global type onto the rank of the communicator (principal of the session) of each role and then type-check the communicator of that role with the role type returned from projection. In contrast to SJ, there is no overhead related to communication-safety at session initiation; i.e., no reciprocity property is checked between the processes of a session at runtime. Our prediction is that parameterised sessions will offer better support for massive parallelism than the SJ client-server based session sockets.

We plan to study and implement parallel algorithms over other unexplored communication patterns, e.g. the Fast Fourier Transformation algorithm over a hypercube pattern previously studied by Yoshida et al. (2010), to further evaluate the expressivity of this system. We also wish to compare the implementation to shared memory communication libraries such as the Partitioned Global Address Space (PGAS) language X10[2] using parallel algorithms as a basis; PGAS is a parallel programming model where local memory is logically allocated to each process from a global address space.

5.2.2 Extending CSMS and CPS

There are several ways to extend this work from a theoretical perspective. We list them in the following paragraphs.

Global types. The global types in CSMS can be written without channels and a procedure, implementing the logic of linearity, can generate them safely and economically; i.e., the same channel can be used in different communications provided that it does not break causalities at runtime. In CPS, the index expressions in principals can be extended to richer mathematical operations such as congruence and a more expressive form of exponentiation, preserving the decidability of the type system.

Enriching the typing relation. Mostrous et al. (2009) enrich the typing discipline of multiparty session types to allow correct programs that specify a different order of "send"s from the global types for optimisation purposes as explained in Section 4.1.2. The subtyping relation on the end-point types $k!\langle U\rangle; k'!\langle U'\rangle; T \ll k'!\langle U'\rangle; k!\langle U\rangle; T$ can be applied to the role types of CPS as $!\langle \mathtt{p}, U\rangle!\langle \mathtt{p}', U'\rangle; T \ll !\langle \mathtt{p}', U'\rangle; !\langle \mathtt{p}, U\rangle; T$ and the subtyping rule in the typing rules will be the same as the one of that system. This technicality is important for a safe library that is used to implement parallel algorithms where optimisations of swapping sending actions are very present.

Delegation. A process in CSMS sends the entire set of session channels when delegating its behaviour, although a part of them is not used to perform that behaviour. Implementing delegation in a programming language carries a cost. Indeed, SJ requires a nontrivial

[2]The web page of X10 can be found in the address `http://x10-lang.com`.

communication protocol to support delegation safely for binary sessions where a single channel is transmitted. An efficient implementation of multiparty session types must allow only the transmission of the channels used in the actions of the delegated behaviour, preserving the type-safety of the type system. Thus, a theory that supports that form of delegation contributes to a more efficient system of multiparty session types.

Multithreading. The progress on local computing mainly due to the introduction of the multicore technology (Borkar et al. (2005)) has followed the progress in global computing due to the increase of Internet services. Present-day single core platforms are able to handle only a few threads but multi-core ones will be able to handle hundreds and in some platforms even thousands of them. As mentioned at the beginning of Section 2.6, CSMS does not support multithreading and therefore cannot benefit from its advantage, raising a short, deep question: How to express multithreading in CSMS, preserving a lightweight, intuitive type annotation and efficient type-checking. For CPS, the study of multithreading must take place only after the one of CSMS is fully evaluated, following an incremental research approach.

Dynamic constructs. More dynamic concepts such as joining and leaving of participants within a session at different points of its execution are of interest in web services and Cloud management systems. We have studied a system (Bejleri et al. (2010)) to control and coordinate components of a distributed computation in the Cloud that allows dynamic addition of components for shared memory. It is well-known that a key advantage of shared memory from message-passing is extensibility— new components can be added dynamically without changing the data flow of the system. This presents the key challenge of providing joining and leaving for mobile processes. We plan to overcome this challenge by describing the examples studied in that system in CPS and so, have a better understanding of the expressivity features missing from our system, regarding addition of participants within a session. Subsequently, we will extend the formal model of CPS with dynamic features by preserving its declarative definition. Very recently, Deniélou and Yoshida (2011) have developed a system where participants can join and leave a session in the context of multiparty session types. Their formal model extends the one of *Honda et al.* through universal quantifiers. However, the study does not show evidence of expressivity differences with parameterised session types.

Appendix A

Synchronous Multiparty Session Types

A.1 Proof of Proposition 2.6.12

The proof of (2) induces concrete algorithms. Global types are generally treated as regular trees, except when we consider substitution. We give the following notation.

Proof of (2) By (1), it suffices to validate the linearity of $G(1)$ to unfold once. We consider the following algorithm.

Input: 0-time unfolded **G** and an empty vector to store the scope of recursion
Output: one-time unfolded **G**

```
procedure one-time-unfolding(G, Scopes)
if G.value is RecursiveNode then
    Scopes.add(G.var, G.clone());
    one-time-unfolding(G.child, Scopes);
else if G.value is VarNode then
    G = Scopes.find(G.child.var);
else
    one-time-unfolding(G.child, Scopes);
```

Algorithm 1: Unfolds a global type one-time.

The elementary object for the above algorithm and the others in Appendix A is the node that stores the prefix and the II, IO, OI and OO dependencies or a recursive variable.

Definition A.1.1. *The size of a global type G is defined as the number of elementary*

objects:

$$size(G) = \begin{cases} |\tilde{k}| + size(G') & \text{if } G = p \to \tilde{p}' : \tilde{k}\,\langle U \rangle.G' \text{ and } U \neq \langle G'' \rangle \\ |\tilde{k}| + size(G') + size(G'') & \text{if } G = p \to \tilde{p}' : \tilde{k}\,\langle U \rangle.G' \text{ and } U = \langle G'' \rangle \\ |\tilde{k}| + \sum_{i \in I} size(G_i) & \text{if } G = p \to \tilde{p}' : \tilde{k}\,\{l_i : G_i\}_{i \in I} \\ 1 + size(G') & \text{if } G = \mu \mathbf{t}.G' \\ 0 & \text{if } G = \mathbf{t} \text{ or } G = \mathsf{end} \end{cases}$$

Given the size of the input $n = size(G)$, the time complexity of the one-time-unfolding algorithm, $\mathrm{T}_{one-time-unfolding}$, is $O(n + d * t)$, where d is the number of recursive definitions and t is the number of leafs that contain a variable. The term $d * t$ defines the number of steps to search in the Scope vector of length d as many time as the number (t) of leafs that contain a variable. In the best case where the input does not contain recursive definitions, the time complexity is $O(n)$; in the worst case, where the number of recursive definitions defines the size of the global type, the time complexity is $O(n^2)$. In the medium case, this algorithms performs better than the one that performs a substitution routine every time it encounter a recursive definition. The time complexity of the latter is defined as $O(n + d * n)$.

The space complexity of the algorithm, $\mathrm{S}_{one-time-unfolding}$, is $O(d * n)$— the number of objects (prefixes) stored during execution. This is because a clone of a global type stores $O(n)$ objects and is invoked as many time as the number of recursive definitions, d. In the best case, the space complexity is $O(1)$ and in the worst case is $O(n^2)$. Below, we discuss the correctness of the algorithm.

Proposition A.1.2 (Partial Correctness)**.** *Given a well-formed global type G and an empty vector then the algorithm returns a well-formed global type G' and a vector containing the scope of each recursive variable where the scope is a well-formed global type.*

Proof. Suppose that $\emptyset \vdash G$ and vector *Scopes* is empty as the algorithm begins then only either the first or the third case of the conditional in the algorithm is satisfied. For the first case, the post condition holds as $var \vdash G.var$ and *Scopes* contains the scope of var: G— a well-formed global type. For the third case, $\vdash G.child$ and empty *Scopes*; thus, the post-conditions are the same to the pre-conditions. We have to proof the inductive case (the recursive call) for the first case as for the third case, follows immediately by induction. The precondition for the inductive case of the first case is $var \vdash G.var$ and *Scopes* contains the scope of var. One of the three cases of the conditional can be satisfied. Since the

first and third case follow immediately by induction, we give the proof for the second case which terminates the algorithm. Since the vector *Scopes* maintains well-formed global types then the global type assigned is well-formed. Also *Scopes* is the same as in the precondition: contains the scope of each recursive variable that is well-formed global type. There is no inductive case for the second case; thus, we conclude. □

Proposition A.1.3 (Termination)**.** *If G is well-formed then each recursive call, given a vector Scopes, will return that vector containing the scopes for each recursive variable.*

Proof. The visit of the global type is finite as the length of a global type is finite, as discussed above, and each recursive call in the first and third case of the conditional advances on each causality or recursive definition. Since the global type is well-formed, then the third case of the global type will encounter only global types having a causality.

□

Theorem A.1.4 (Correctness of One-time-unfolded)**.** *Given a global type Gthen the algorithm returns the one-time unfolding of it.*

Proof. Follows immediately from Propositions A.1.2 and A.1.3.

□

The algorithm that constructs the dependencies between the prefixes of a global type is given below.

Input: 1-time unfolded global type **G**
Output: **G** extended with order dependencies

procedure construct_dependencies(G)
construct_dependenciesNode(G, G.child);
construct_dependencies(G.child);

Algorithm 2: Builds dependencies between nodes of a global type.

The algorithm builds the dependencies recursively on each node of the tree.

Given the size of the input $m = size(G')$, the time complexity of construct_dependencyNode is $O(m)$— the number of steps to traverse G'. Given the size of the input $n = size(G)$, the time complexity of construct_dependencies is:

$$O(n) \leq T_{construct_dependencies} \leq O(n^2)$$

Input: Node G and global type G′
Output: **true** G extended with dependencies of nodes of G′

procedure construct_dependenciesNode(G, G')
if $G \prec_{\mathtt{II}} G'$ **then**
  addII(G, G');
else if $G \prec_{\mathtt{IO}} G'$ **then**
  addIO(G, G');
else if $G \prec_{\mathtt{OO}} G'$ **then**
  addOO(G,G');
else if $G \prec_{\mathtt{OI}} G'$ **then**
  addOI(G, G');
else Error;
construct_dependenciesNode(G, G'.child);

Algorithm 3: Builds dependencies for a node.

where the lower bounder is defined in the case when G is a branching node and the size of G is the number of branches, and the upper bounder is defined in the case when G does not contain branching construct.

The space complexity of construct_dependencies is:

$$O(1) \leq S_{construct_dependencies} \leq O(n^2)$$

where the lower bound represents the best case of no dependencies, and the upper bound represents the worst case of dependencies for each node (as many as the size of the subtree that follows). Below, we discuss the correctness of the algorithm.

Theorem A.1.5 (Correctness of Construct_Dependencies)**.** *Given a global type G then the algorithm returns G extended with the dependencies on each causality.*

Proof. Suppose that $\emptyset \vdash G$ as the algorithm begins then *construct_dependenciesNode* returns G with dependencies of the first causality. By induction, *construct_dependenciesNode* will return G with the dependencies on the following causalities. Partial correctness of *construct_dependenciesNode* follows immediately after a pre and post-condition analysis, where the pre-condition defines the first argument as well-formed causality and the post-condition defines the same causality with or without a dependency, and well-formed. Recalling from Chapter 2, a global type can be represented as a tree; thus, the termination *construct_dependenciesNode* and *construct_dependencies* follows immediately.

□

Linearity checks for both input and output chains in all communications of the same channel. Given $n = size(G)$, the time complexity of linearity is in the best case, where no channel is used more than one time, is $O(n)$. The complexity is defined by the number of steps to visit G. Otherwise, the time complexity is defined as a search over the nodes for an input and output dependency. The time complexity of input_dependency is $O(n^2)$. The search is linear $O(n)$ and is repeated for each node of a global type. The time complexity of output_dependency is quadratic $O(n^2)$, which defines the number of steps to perform a linear search on the vector of prefixes, every time a node is reached by the visit of a global type. In the worst case when half of the graph node channels $(n/2)$ are used for two times then the complexity of linearity is $O(n^3)$.

```
Input: 1-time unfolded global type G with dependencies and empty vector of
       prefixes indexed by channel number
Output: true if G is linear, false otherwise

boolean linearity(G, Prefixes)
if Prefixes[G.channel] ≠ null then
    if ( input_chain(Prefixes[G.channel], G, new Vector()) and
    output_chain(Prefixes[G.channel], G, new Vector()) ) then
        Prefixes[G.channel] = G;
        if G.value is BranchingNode then
            foreach i ∈ G.child do
                if !(linearity(G.child(i), clone(Prefixes))) then
                    return false ;
            return true
        else
            return linearity(G.child, Prefixes);
    else
        return false ;
else
    Prefixes[G.channel] = G;
    return linearity(G.child, Prefixes);
```

Algorithm 4: Checks linearity.

The space complexity of linearity in the best case is $O(1)$ and in in the worst case is

$O(n^2)$, where the size of the Prefix array is $O(n)$ and n times it might be copied.

Input: Node G and G' to look for input chain

Output: **true** if the chain exits, **false** otherwise

```
boolean input_chain(G, G')
if G = G' then
    return true;
else if G' ≺ G then
    return false;
else if G≺II G' or G≺OI G' then
    return true;
else  foreach i ∈ IO, OI, II, OO(G) do
    if i=OI or II then
        if i.child = OO or OI then
            input_chain(i.child, G');
        else
            return false ;
    else if i=IO or OO then
        if i.child = II or IO then
            input_chain(i.child, G');
        else
            return false ;
    else
        error;
```

Algorithm 5: Checks for input chain.

Input: Node G and G' to look for output chain
Output: **true** if the chain exits, **false** otherwise

```
boolean output_chain(G, G')
if G = G' then
    return true;
else if G' ≺ G then
    return false;
else if G≺_OO G' or G≺_IO G' then
    return true;
else  foreach i ∈ IO, OI, II, OO(G) do
    if i=OI or II then
        if i.child = OO or OI then
            output_chain(i.child, G');
        else
            return false ;
    else if i=IO or OO then
        if i.child = II or IO then
            output_chain(i.child, G');
        else
            return false ;
    else
        error;
```

Algorithm 6: Checks for output chain.

Conjecture A.1.6 (Correctness of Linearity)**.** *Given a global type G with dependencies on each causality then the algorithm returns* ***false*** *if for two causalities using the same channel either input_channel or output_chain return* ***false****, otherwise returns* ***true*** *.*

Termination of the algorithm follows immediately by the fact that a global type can be represented as a tree and the algorithm is a visit of that tree. Partial correctness follows by the pre-condition that captures a global type extended with dependencies and a vector of Prefixes indexed by a channel, containing the last causality that use that channel during the visit. The post-condition captures *input_chain* and *output_channel* returning true if the channel of the causality in the precondition is not an element of Prefixes, or is an element and there is an input and output path (defined in Section 2.6.3) between the causality in the precondition and the one in Prefixes (the latter assertion follows from the

partial correctness of *input_chain* and *output_chain*); otherwise returns false.

A.2 Proof of Theorem 2.6.18

This section presents the algorithm that defines the coherence of a global type and, its time and space complexity.

Input: Global type **G**
Output: **true** if **G** is coherent, **false** otherwise

boolean coherence(G)
G′ = clone(G);
one-time-unfolded(G′);
construct_dependencies(G′);
return linearity(G′, new PrefixesVector) **and** projection(G, new End-pointVector);

Algorithm 7: Checks coherence of a global type.

Before defining the time and space complexity of coherence, we discuss the computational complexity of projection. Given $n = size(G)$,

$$O(n) \leq T_{projection} \leq O(n * logn)$$

where the term $O(n)$ is defined in the best case where there are no branching, and the term $O(n * logn)$ is defined in the worst case where the number of branchings defines the size of the input. For each subtree, the algorithm checks if the end-point types are mergeable The length of subtrees is logarithmic to the number of nodes, where the base of logarithm is defined by the number of branches. The number of subtrees is the length of the input , $O(n)$. Space complexity is $O(1)$ in the best case and in the worst case is $O(n)$.

Given $n = size(G)$, the time complexity of coherence is the addition of the time complexity of one-time-unfolding, construct_dependencies, linearity and projection. In the best case, where there are no recursive definitions, $T_{one-time-unfolding} = O(n)$. The time complexity of construct_dependencies is always quadratic on the input, therefore in the best case is $O(n^2)$. In the best case, where there are no multiple use of channels, $T_{linearity} = O(n)$ and where there are no branching , the time complexity of coherence is defined by the complexity factor of construct_dependencies (n^2). In the worst case, $T_{one-time-unfolding} = O(n^2)$ and the space complexity is $O(n^2)$; given the size of G' $O(n)$

then construct_dependencies is $O(n^2)$. The time complexity of linearity is $O(n^3)$. The time complexity of projection is $O(n * logn)$ and the time complexity of coherence is defined by the complexity factor of linearity, $O(n^3)$. The space complexity is defined by the complexity factor of construct_dependencies $O(n^2)$.

```
Input: Global type G and an array of end-points indexed by participant
Output: true if G is coherent, false otherwise with the vector of End-point Types

boolean projection(G, End-points)
if G.value is ValueNode then
    End-points[G.firstParticipant].addSend(G.channel, G.value);
    foreach i ∈ G.secondParticipant do
        End-points[i].addRecv(G.channel(i), G.value);
    return projection(G.child, End-points) & projection(G.message, new
    End-pointVector);
else if G.value is BranchNode then
    temp = new End-pointVector;
    foreach i ∈ G.child do
        if projection(G.child(i), temp) then
            End-points[G.firstParticipant].addSelection(G.channel, G.label(i),
            temp[G.firstParticipant]);
            foreach j ∈ G.secondParticipant do
                End-points[j].addBranching(G.channel(i), G.label(i), temp[j]);
            foreach j ∈ temp & (j ≠ G.firstParticipant, G.secondParticipant) do
                End-points[j].add(temp[j]);
        else
            return false ;
    return mergeability(End-points, G.firstParticipant, G.secondParticipant);
else if G is RecursiveNode then
    foreach el ∈ End-points do
        End-points[el].addRecursion(G.value);
    return projection(G.child, End-points);
else  foreach el ∈ End-points do
    End-Points[el].addLeaf(G.value);
    return true ;
```

Algorithm 8: Projection of global types.

Conjecture A.2.1 (Correctness of Projection)**.** *Given a global type G, the coherence algorithm returns* ***true*** *if all branches of G are mergeable,* ***false*** *otherwise.*

Conjecture A.2.2 (Correctness of Coherence)**.** *Given a global type G, the coherence algorithm returns* ***true*** *if G is linear and projectable,* ***false*** *otherwise.*

The correctness of the algorithm follows from the correctness of the algorithms: $one - time - unfolded, construct_dependencies, linearity$ and $projection$.

Appendix B

Parameterised Session Types

B.1 Auxiliary Definitions

In this section, we give the definition of principals in a global type (Figure B.1), free index variable of the principal that performs each action on role types (Figure B.2) and free index variables in principals (Figure B.3).

$$\mathsf{pid}(\mathtt{p} \to \mathtt{p}' : \langle U \rangle . G) = \{\mathtt{p}, \mathtt{p}'\} \cup \mathsf{pid}(G)$$
$$\mathsf{pid}(\mathtt{p} \to \mathtt{p}' : \{l_i : G_i\}_{i \in I}) = \{\mathtt{p}, \mathtt{p}'\} \cup \bigcup_{i \in I} \mathsf{pid}(G_i)$$
$$\mathsf{pid}(\mu \mathbf{t}.G) = \mathsf{pid}(G) \quad \mathsf{pid}(\mathbf{t}) = \emptyset$$
$$\mathsf{pid}(\mathbf{R}\ G\ \lambda i.\lambda \mathbf{x}.G') = \mathsf{pid}(G) \cup \mathsf{pid}(G')$$
$$\mathsf{pid}(G\ \mathtt{t}) = \mathsf{pid}(G) \quad \mathsf{pid}(\mathsf{end}) = \emptyset \quad \mathsf{pid}(\mathbf{x}) = \emptyset$$

Figure B.1: CPS: Principal identifiers of a global type.

$$\mathrm{fivr}(!\langle \mathtt{p}, U \rangle(\mathtt{p}'); T) = \mathsf{fiv}(\mathtt{p}') \cup \mathrm{fivr}(T)$$
$$\mathrm{fivr}(?\langle \mathtt{p}, U \rangle(\mathtt{p}'); T) = \mathsf{fiv}(\mathtt{p}') \cup \mathrm{fivr}(T)$$
$$\mathrm{fivr}(\oplus\langle \mathtt{p}, \{_i : T_i\}_{ß \in I} \rangle(\mathtt{p}')) = \mathsf{fiv}(\mathtt{p}') \bigcup_{i \in I} \mathrm{fivr}(T_i)$$
$$\mathrm{fivr}(\&\langle \mathtt{p}, \{_i : T_i\}_{ß \in I} \rangle(\mathtt{p}')) = \mathsf{fiv}(\mathtt{p}') \bigcup_{i \in I} \mathrm{fivr}(T_i)$$
$$\mathrm{fivr}(\mu \mathbf{t}.T) = \mathrm{fivr}(T) \quad \mathrm{fivr}(\mathbf{t}) = \emptyset$$
$$\mathrm{fivr}(\mathbf{R}\ T\ \lambda i.\lambda x.T') = \mathrm{fivr}(T) \cup \mathrm{fivr}(T') \setminus \{i\}$$
$$\mathrm{fivr}(T\ \mathtt{t}) = \mathrm{fivr}(T) \quad \mathrm{fivr}(\mathbf{x}) = \emptyset \quad \mathrm{fivr}(\mathsf{end}) = \emptyset$$

Figure B.2: CPS: Free index variables of the principal that perform each action of roles.

$$\mathsf{fiv}(\mathcal{N}) = \emptyset$$
$$\mathsf{fiv}(\mathtt{p}[\mathtt{i}]) = \mathsf{fiv}(\mathtt{i}) \cup \mathsf{fiv}(\mathtt{p})$$
$$\mathsf{fiv}(\mathtt{t}) = \emptyset$$
$$\mathsf{fiv}(i) = \{i\}$$
$$\mathsf{fiv}(\mathrm{n} * \mathtt{i}) = \mathsf{fiv}(\mathtt{i})$$
$$\mathsf{fiv}(\mathtt{t} \pm \mathtt{i}) = \mathsf{fiv}(\mathtt{i})$$

Figure B.3: CPS: Free index variables of principals.

B.2 Examples

B.2.1 2D-Mesh

This section gives the roles and the main program of the 2D-Mesh given in Section 3.2.3. To ensure the presence of all nine roles, we set the parameter domain to $\mathrm{n}, \mathrm{m} \geq 2$.

$\mathtt{p} \triangleq \mathbf{R}\ (\mathbf{R}\ \emptyset\ \lambda k.\lambda Z.\mathtt{W}[n][k]\ m-1)$
$\lambda i.\lambda X.\mathbf{R}\ X$
$\lambda j.\lambda Y.\mathtt{W}[i][j], Y$
$m+1$
n

$$\begin{aligned}
P_{\text{start}}(n,m) &= \bar{a}[\mathtt{W}[n][m], \mathtt{p}](y).y!\langle \mathtt{W}[n-1][m], f(n-1,m)\rangle; \\
&\quad y!\langle \mathtt{W}[n][m-1], f(n,m-1)\rangle; \mathbf{0} \\
P_{\text{top_right_corner}}(n) &= a[\mathtt{W}[n][0]](y).y?(\mathtt{W}[n][1], z); y!\langle \mathtt{W}[n-1][0], f(n-1,0)\rangle; \mathbf{0} \\
P_{\text{bottom_left_corner}}(m) &= a[\mathtt{W}[0][m]](y).y?(\mathtt{W}[1][m], z); y!\langle \mathtt{W}[0][m-1], f(0,m-1)\rangle; \mathbf{0} \\
P_{\text{bottom_right_corner}}(m) &= a[\mathtt{W}[0][0]](y).y?(\mathtt{W}[1][0], z_1); y?(\mathtt{W}[0][1], z_2); \mathbf{0} \\
P_{\text{top_middle}}(n,k) &= a[\mathtt{W}[n][k+1]](y).y?(\mathtt{W}[\mathtt{n}][\mathtt{k}+2], z_1); \\
&\quad y!\langle \mathtt{W}[\mathtt{n}-1][\mathtt{k}+1], f(n-1,k+1)\rangle; y!\langle \mathtt{W}[n][k], f(n,k)\rangle; \mathbf{0} \\
P_{\text{bottom_middle}}(k) &= a[\mathtt{W}[0][k+1]](y).y?(\mathtt{W}[1][k+1], z_1); y?(\mathtt{W}[0][k+2], z_2); \\
&\quad y!\langle \mathtt{W}[0][k], f(0,k)\rangle; \mathbf{0} \\
P_{\text{left_middle}}(m,i) &= a[\mathtt{W}[i+1][m]](y).y?(\mathtt{W}[i+2][m], z_1); y!\langle \mathtt{W}[i][m], f(i,m)\rangle; \\
&\quad y!\langle \mathtt{W}[i+1][m-1], f(i+1,m-1)\rangle; \\
P_{\text{right_middle}}(i) &= a[\mathtt{W}[i+1][0]](y).y?(\mathtt{W}[i+2][0], z_1); y?(\mathtt{W}[i+1][1], z_2); \\
&\quad y!\langle \mathtt{W}[i][0], f(i,0)\rangle; \mathbf{0} \\
P_{\text{center}}(i,j) &= a[\mathtt{W}[i+1][j+1]](y).y?(\mathtt{W}[i+2][j+1], z_1); y?(\mathtt{W}[i+1][j+2], z_2); \\
&\quad y!\langle \mathtt{W}[i][j+1], f(i,j+1)\rangle; y!\langle \mathtt{W}[i+1][j], f(i+1,j)\rangle; \mathbf{0}
\end{aligned}$$

$$\begin{array}{l}\lambda n.\lambda m.(\mathbf{R}\ (\mathbf{R}\ P_{\text{start}}(n,m)|P_{\text{bottom_right_corner}}(m)|P_{\text{top_right_corner}}(n)|P_{\text{bottom_left_corner}}(m))\\ \qquad\qquad \lambda k.\lambda Z.(P_{\text{top_row}}(n,k)|P_{\text{bottom_middle}}(k)|Z)\\ \qquad\qquad m-1)\\ \qquad\qquad \lambda i.\lambda X.(\mathbf{R}\ P_{\text{left_middle}}(m,i)|P_{\text{right_middle}}(i)|X\\ \qquad\qquad\qquad\qquad \lambda j.\lambda Y.(P_{\text{center}}(i,j)|Y)\\ \qquad\qquad\qquad\qquad m-1)\\ \qquad\qquad n-1)\end{array}$$

B.2.2 Jacobi Solution of the Poisson Equation

This section presents the implementation of the roles and main program of the Jacobi solution given in Section 3.3.1.

$$\begin{array}{l}\lambda n.\lambda m.\mathbf{R}\ P_{\text{start}}(n,m) \mid P_{\text{RBC}}(n) \mid P_{\text{LTC}}(m) \mid P_{\text{LBC}}\\ \qquad\qquad \lambda i.\lambda X.\mathbf{R}\ (P_{\text{LC}}(i) \mid P_{\text{RC}}(i) \mid X)\lambda j.\lambda Y.(P_{\text{centre}}(i,j) \mid P_{\text{TR}}(j) \mid P_{\text{BR}}(j) \mid Y)\ n\ m\end{array}$$

$$\begin{array}{rl}P_{\text{centre}}(i,j) & = a[\mathtt{W}[i][j]](y).\mu\mathbf{t}.y\&\langle \mathtt{W}[i][j+1], \{true : y \oplus \langle \mathtt{W}[i][j-1]\{true :\\ & \qquad y?(\mathtt{W}[i+1][j], z_1); y!\langle \mathtt{W}[i+1][j], f(i+1,j)\rangle;\\ & \qquad y?(\mathtt{W}[i][j+1], z_2); y!\langle \mathtt{W}[i][j+1], f(i,j+1)\rangle;\\ & \qquad y?(\mathtt{W}[i][j-1], z_3); y!\langle \mathtt{W}[i][j-1], f(i,j-1)\rangle;\\ & \qquad y?(\mathtt{W}[i-1][j], z_4); y!\langle \mathtt{W}[i-1][j], f(i-1,j)\rangle;\\ & \qquad y?(\mathtt{W}[i][j-1], z_5); y!\langle \mathtt{W}[i][j+1], f'(z_5)\rangle; \mathbf{t}\}\rangle,\\ & \qquad\qquad false : y \oplus \langle \mathtt{W}[i][j-1], \{false :\\ & \qquad y?(\mathtt{W}[i][j-1], z_6); y!\langle \mathtt{W}[i][j+1], f''(z_6)\rangle; \mathbf{0}\}\rangle\}\rangle\\ P_{\text{start}}(n,m) & = \bar{a}[\mathtt{W}[0][0]..\mathtt{W}[n][m]](y).\mu\mathbf{t}.y \oplus \langle \mathtt{W}[n][m-1], \mathtt{W}[n-1][m]\{true :\\ & \qquad y!\langle \mathtt{W}[n][m-1], f(n,m-1)\rangle; y?(\mathtt{W}[n][m-1], z);\\ & \qquad y!\langle \mathtt{W}[n-1][m], f(n-1,m)\rangle; y?(\mathtt{W}[n-1][m], z_2);\\ & \qquad y?(\mathtt{W}[n-1][m], z_3); y?(\mathtt{W}[n][m-1], z_4)); \mathbf{t},\\ & \qquad\qquad false : y?(\mathtt{W}[n-1][m], z_5); y?(\mathtt{W}[n][m-1], z_6); \mathbf{0}\}\rangle\\ P_{\text{LBC}}(m) & = a[\mathtt{W}[0][m]](y).\mu\mathbf{t}.y\&\langle \mathtt{W}[1][m], \{true : y \oplus \langle \mathtt{W}[0][m-1]\{true :\\ & \qquad y?(\mathtt{W}[1][m], z_1); y!\langle \mathtt{W}[1][m], f(i+1,j)\rangle;\\ & \qquad y!\langle \mathtt{W}[0][m-1], f(0,1)\rangle; y?(\mathtt{W}[0][m-1], z_2);\\ & \qquad y?(\mathtt{W}[0][m-1], z_5); y!\langle \mathtt{W}[1][m], f'(z_5)\rangle; \mathbf{t}\}\rangle,\\ & \qquad\qquad false : y \oplus \langle \mathtt{W}[1][m], \{false :\\ & \qquad y?(\mathtt{W}[0]m-1], z_6); y!\langle \mathtt{W}[1][m], f''(z_6)\rangle; \mathbf{0}\}\rangle\}\rangle\end{array}$$

$$
\begin{aligned}
P_{\mathrm{TR}}(j) = a[\mathtt{W}[n][j]](y).\mu\mathbf{t}.y\&\langle \mathtt{W}[n][j+1], \{true : y \oplus \langle \mathtt{W}[n][j-1]\{true :\\
& y?(\mathtt{W}[n][j+1], z_2); y!\langle \mathtt{W}[n][j+1], f(i, j+1)\rangle;\\
& y?(\mathtt{W}[n][j-1], z_3); y!\langle \mathtt{W}[n][j-1], f(i, j-1)\rangle;\\
& y?(\mathtt{W}[n-1][j], z_4); y!\langle \mathtt{W}[n-1][j], f(i-1, j)\rangle;\\
& y?(\mathtt{W}[n][j-1], z_5); y!\langle \mathtt{W}[n][j+1], f'(z_5)\rangle; \mathbf{t}\}\rangle,\\
& \qquad\qquad\qquad\qquad false : y \oplus \langle \mathtt{W}[n][j-1], \{false :\\
& y?(\mathtt{W}[n][j-1], z_6); y!\langle \mathtt{W}[n][j+1], f''(z_6)\rangle; \mathbf{0}\}\rangle\}\rangle
\end{aligned}
$$

The implementation of the other roles is similar and can be easily created from the definition of P_{center}.

References

Alves, S., Fernández, M., Florido, M., and Mackie, I. (**2010**). "Godel's system T revisited." *Theor. Comput. Sci.*, 411: 1484–1500.

Amtoft, T., Nielson, F., and Nielson, H. R. (**1997**). "Type and Behaviour Reconstruction for Higher-Order Concurrent Programs." *J. Funct. Program.*, 7(3): 321–347.

Ateniese, G., Steiner, M., and Tsudik, G. (**1998**). "Authenticated group key agreement and friends." In "Proceedings of the 5th ACM conference on Computer and Communications Security," CCS '98, pages 17–26. ACM, New York, NY, USA.

Bejleri, A., Farrell, A., and Goldsack, P. (**2010**). "Cloudscape: Language Support to Coordinate and Control Distributed Applications in the Cloud." Technical report, Hewlett-Packard Labs.

Bejleri, A., Hu, R., and Yoshida, N. (**2009**). "Session-Based Programming for Parallel Algorithms: Expressiveness and Performance." In A. R. Beresford and S. J. Gay, editors, "PLACES," volume 17 of *EPTCS*, pages 17–29.

Bettini, L., Coppo, M., D'Antoni, L., Luca, M. D., Dezani-Ciancaglini, M., and Yoshida, N. (**2008**). "Global Progress in Dynamically Interleaved Multiparty Sessions." In F. van Breugel and M. Chechik, editors, "CONCUR," volume 5201 of *Lecture Notes in Computer Science*, pages 418–433. Springer.

Bonelli, E. and Compagnoni, A. B. (**2007**). "Multipoint Session Types for a Distributed Calculus." In G. Barthe and C. Fournet, editors, "TGC," volume 4912 of *Lecture Notes in Computer Science*, pages 240–256. Springer.

Borkar, S., Dubey, P., Kahn, K., Kuck, D., Mulder, H., Pawlowski, S., and Rattner, J. (**2005**). "Platform 2015: Intel Processor and Platform Evolution for the Next Decade." *Technology@Intel Magazine*.

Bracha, G. (**2004**). "Pluggable Type Systems." In "RDL," Proceedings of the OOPSLA Workshop on Revival of Dynamic Languages.

Bruni, R., Lanese, I., Melgratti, H. C., and Tuosto, E. (**2008**). "Multiparty Sessions in SOC." In D. Lea and G. Zavattaro, editors, "COORDINATION," volume 5052 of *Lecture Notes in Computer Science*, pages 67–82. Springer.

Bugliesi, M., Castagna, G., and Crafa, S. (**2004**). "Access control for mobile agents: The calculus of boxed ambients." *ACM Trans. Program. Lang. Syst.*, 26(1): 57–124.

Caires, L. and Vieira, H. T. (**2010**). "Conversation types." *Theor. Comput. Sci.*, 411(51-52): 4399–4440.

Carbone, M., Honda, K., and Yoshida, N. (**2007**a). "A Calculus of Global Interaction based on Session Types." *Electr. Notes Theor. Comput. Sci.*, 171(3): 127–151.

Carbone, M., Honda, K., and Yoshida, N. (**2007**b). "Structured Communication-Centred Programming for Web Services." In R. D. Nicola, editor, "ESOP," volume 4421 of *Lecture Notes in Computer Science*, pages 2–17. Springer.

Carbone, M., Yoshida, N., and Honda, K. (**2009**). "Asynchronous Session Types: Exceptions and Multiparty Interactions." In M. Bernardo, L. Padovani, and G. Zavattaro, editors, "SFM," volume 5569 of *Lecture Notes in Computer Science*, pages 187–212. Springer.

Castagna, G., Gesbert, N., and Padovani, L. (**2009**). "A theory of contracts for Web services." *ACM Trans. Program. Lang. Syst.*, 31(5).

Castagna, G. and Padovani, L. (**2009**). "Contracts for Mobile Processes." In M. Bravetti and G. Zavattaro, editors, "CONCUR," volume 5710 of *Lecture Notes in Computer Science*, pages 211–228. Springer.

Clarke, D. G., Potter, J. M., and Noble, J. (**1998**). "Ownership types for flexible alias protection." In "Proceedings of the 13th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications," OOPSLA '98, pages 48–64. ACM, New York, NY, USA.

Coppo, M., Dezani-Ciancaglini, M., and Yoshida, N. (**2007**). "Asynchronous Session Types and Progress for Object Oriented Languages." In M. M. Bonsangue and

E. B. Johnsen, editors, "FMOODS," volume 4468 of *Lecture Notes in Computer Science*, pages 1–31. Springer.

Corin, R., Denielou, P.-M., Fournet, C., Bhargavan, K., and Leifer, J. (**2007**). "Secure Implementations for Typed Session Abstractions." In "Proceedings of the 20th IEEE Computer Security Foundations Symposium," pages 170–186. IEEE Computer Society, Washington, DC, USA.

DeLine, R. and Fähndrich, M. (**2004**). "Typestates for Objects." In M. Odersky, editor, "ECOOP," volume 3086 of *Lecture Notes in Computer Science*, pages 465–490. Springer.

Deniélou, P.-M. and Yoshida, N. (**2011**). "Dynamic multirole session types." In T. Ball and M. Sagiv, editors, "POPL," pages 435–446. ACM.

Dezani-Ciancaglini, M., de'Liguoro, U., and Yoshida, N. (**2007**). "On Progress for Structured Communications." In G. Barthe and C. Fournet, editors, "TGC," volume 4912 of *Lecture Notes in Computer Science*, pages 257–275. Springer.

Dezani-Ciancaglini, M., Mostrous, D., Yoshida, N., and Drossopoulou, S. (**2006**). "Session Types for Object-Oriented Languages." In D. Thomas, editor, "ECOOP," volume 4067 of *Lecture Notes in Computer Science*, pages 328–352. Springer.

Fähndrich, M., Aiken, M., Hawblitzel, C., Hodson, O., Hunt, G. C., Larus, J. R., and Levi, S. (**2006**). "Language support for fast and reliable message-based communication in singularity OS." In Y. Berbers and W. Zwaenepoel, editors, "EuroSys," pages 177–190. ACM.

Fowler, M. and Scott, K. (**1997**). *UML distilled - applying the standard object modeling language*. Addison-Wesley-Longman.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (**1995**). *Design Patterns: Elements of Reusable Design*. Addison Wesley.

Garralda, P., Compagnoni, A., and Dezani-Ciancaglini, M. (**2006**). "BASS: boxed ambients with safe sessions." In "Proceedings of the 8th ACM SIGPLAN international conference on Principles and practice of declarative programming," PPDP '06, pages 61–72. ACM, New York, NY, USA.

Gay, S. J. (**2008**). "Bounded polymorphism in session types." *Mathematical Structures in Computer Science*, 18(5): 895–930.

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

Gay, S. J., Vasconcelos, V. T., Ravara, A., Gesbert, N., and Caldeira, A. Z. (**2010**). "Modular session types for distributed object-oriented programming." In M. V. Hermenegildo and J. Palsberg, editors, "POPL," pages 299–312. ACM.

Giachino, E., Sackman, M., Drossopoulou, S., and Eisenbach, S. (**2009**). "Softly safely spoken: Role playing for Session Types." In "PLACES'09," Proceedings of the ETAPS Workshop on Programming Language Approaches to Concurrency and Communication-cEntric Software.

Girard, J.-Y. (**1987**). "Linear Logic." *Theor. Comput. Sci.*, 50: 1–102.

Gropp, W., Lusk, E., and Skjellum, A. (**1999**). *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. MIT Press.

Grosso, W. (**2001**). *Java RMI*. O'Reilly Media.

Honda, K., Vasconcelos, V. T., and Kubo, M. (**1998**). "Language Primitives and Type Discipline for Structured Communication-Based Programming." In C. Hankin, editor, "ESOP," volume 1381 of *Lecture Notes in Computer Science*, pages 122–138. Springer.

Honda, K., Yoshida, N., Bejleri, A., and Carbone, M. (**2008**a). "Multiparty Asynchronous Session Types." Available at: `www.doc.ic.ac.uk/~yoshida/multiparty`. Typescript, 60 pages (Submitted for publication).

Honda, K., Yoshida, N., and Carbone, M. (**2008**b). "Multiparty asynchronous session types." In G. C. Necula and P. Wadler, editors, "POPL," pages 273–284. ACM.

Hu, R., Yoshida, N., Bejleri, A., and Honda, K. (**2009**). "The SJ Framework for Transport-Independent, Type-Safe, Objet-Oriented Communications Programming." Available at `http://www.doc.ic.ac.uk/~rhu/sessionj-ti.html`.

Hu, R., Yoshida, N., and Honda, K. (**2008**). "Session-Based Distributed Programming in Java." In J. Vitek, editor, "ECOOP," volume 5142 of *Lecture Notes in Computer Science*, pages 516–541. Springer.

Igarashi, A. and Kobayashi, N. (**2004**). "A generic type system for the Pi-calculus." *Theor. Comput. Sci.*, 311(1-3): 121–163.

Kobayashi, N. (**2006**). "A New Type System for Deadlock-Free Processes." In C. Baier and H. Hermanns, editors, "CONCUR," volume 4137 of *Lecture Notes in Computer Science*, pages 233–247. Springer.

Kobayashi, N., Saito, S., and Sumii, E. (**2000**). "An Implicitly-Typed Deadlock-Free Process Calculus." In C. Palamidessi, editor, "CONCUR," volume 1877 of *Lecture Notes in Computer Science*, pages 489–503. Springer.

Lapadula, A., Pugliese, R., and Tiezzi, F. (**2007**). "A Calculus for Orchestration of Web Services." In R. D. Nicola, editor, "ESOP," volume 4421 of *Lecture Notes in Computer Science*, pages 33–47. Springer.

Leighton, F. T. (**1991**). *Introduction to parallel algorithms and architectures: arrays, trees, hypercubes*. Morgan Kaufmann.

Martin-Lőf, P. (**1972**). "Infinite Terms and a System of Natural Deduction." In "Compositio Mathematica," pages 93–103. Wolters-Noordhoof.

Mezzina, L. G. (**2008**). "How to Infer Finite Session Types in a Calculus of Services and Sessions." In D. Lea and G. Zavattaro, editors, "COORDINATION," volume 5052 of *Lecture Notes in Computer Science*, pages 216–231. Springer.

Milner, R., Parrow, J., and Walker, D. (**1992**). "A Calculus of Mobile Processes, Parts I and II." *Information and Computation*, 100(1).

Milner, R., Tofte, M., and Macqueen, D. (**1997**). *The Definition of Standard ML*. MIT Press, Cambridge, MA, USA.

Mostrous, D. and Yoshida, N. (**2007**). "Two Session Typing Systems for Higher-Order Mobile Processes." In S. R. D. Rocca, editor, "TLCA," volume 4583 of *Lecture Notes in Computer Science*, pages 321–335. Springer.

Mostrous, D., Yoshida, N., and Honda, K. (**2009**). "Global Principal Typing in Partially Commutative Asynchronous Sessions." In G. Castagna, editor, "ESOP," volume 5502 of *Lecture Notes in Computer Science*, pages 316–332. Springer.

MSCWG (**1996**). *ITU-TS recommendation Z.120: Message Sequence Chart (MSC)*.

Nelson, N. (**1991**). "Primitive Recursive Functionals with Dependent Types." In S. D. Brookes, M. G. Main, A. Melton, M. W. Mislove, and D. A. Schmidt, editors, "MFPS," volume 598 of *Lecture Notes in Computer Science*, pages 125–143. Springer.

Neubauer, M. and Thiemann, P. (**2004**). "An Implementation of Session Types." In B. Jayaraman, editor, "PADL," volume 3057 of *Lecture Notes in Computer Science*, pages 56–70. Springer.

Nielson, H. R., Amtoft, T., and Nielson, F. (**1998**). "Behaviour Analysis and Safety Conditions: A Case Study in CML." In "FASE," pages 255–269.

Nielson, H. R. and Nielson, F. (**1994**). "Higher-order concurrent programs with finite communication topology (extended abstract)." In "Proceedings of the 21st ACM SIGPLAN-SIGACT symposium on Principles of programming languages," POPL '94, pages 84–97. ACM, New York, NY, USA.

Nystrom, N., Clarkson, M. R., and Myers, A. C. (**2003**). "Polyglot: An Extensible Compiler Framework for Java." In G. Hedin, editor, "CC," volume 2622 of *Lecture Notes in Computer Science*, pages 138–152. Springer.

Pereira, N. S. A. (**1998**). "A Parallel N-Body Integrator Using MPI." In J. M. L. M. Palma, J. Dongarra, and V. Hernández, editors, "VECPAR," volume 1573 of *Lecture Notes in Computer Science*, pages 627–639. Springer.

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

Pucella, R. and Tov, J. A. (**2008**). "Haskell session types with (almost) no class." In "Proceedings of the first ACM SIGPLAN symposium on Haskell," Haskell '08, pages 25–36. ACM, New York, NY, USA.

Pugh, W. and Wonnacott, D. (**1992**). "Eliminating false data dependences using the Omega test." In "Proceedings of the ACM SIGPLAN 1992 conference on Programming language design and implementation," PLDI '92, pages 140–151. ACM, New York, NY, USA.

Reppy, J. H. (**1991**). "CML: A higher concurrent language." In "Proceedings of the ACM SIGPLAN 1991 conference on Programming language design and implementation," PLDI '91, pages 293–305. ACM, New York, NY, USA.

Sackman, M. and Eisenbach, S. (**2008**). "Session Types in Haskell: Updating Message Passing for the 21st Century." Technical report, Imperial College London.

Shafi, A. (**2006**). *Nested Parallelism for Multi-core Systems Using Java*. Ph.D. thesis, University of Portsmouth.

Sparkes, S. (**2006**). "Conversation with Steve Ross-Talbot." *ACM Queue*, 4(2): 14–23.

Spring, J. H., Privat, J., Guerraoui, R., and Vitek, J. (**2007**). "Streamflex: high-throughput stream programming in java." In R. P. Gabriel, D. F. Bacon, C. V. Lopes, and G. L. S. Jr., editors, "OOPSLA," pages 211–228. ACM.

Tait, W. W. (**1965**). "Infinitely Long Terms of Transfinite Type." In Crossley and Dummett, editors, "Formal Systems and Recursive Functions," pages 177–185. North Holland.

Takeuchi, K., Honda, K., and Kubo, M. (**1994**). "An Interaction-based Language and its Typing System." In C. Halatsis, D. G. Maritsas, G. Philokyprou, and S. Theodoridis, editors, "PARLE," volume 817 of *Lecture Notes in Computer Science*, pages 398–413. Springer.

Vasconcelos, V. T., Gay, S. J., and Ravara, A. (**2006**). "Type checking a multithreaded functional language with session types." *Theor. Comput. Sci.*, 368(1-2): 64–87.

Vasconcelos, V. T., Ravara, A., and Gay, S. J. (**2004**). "Session Types for Functional Multithreading." In P. Gardner and N. Yoshida, editors, "CONCUR," volume 3170 of *Lecture Notes in Computer Science*, pages 497–511. Springer.

Vieira, H. T., Caires, L., and Seco, J. C. (**2008**). "The Conversation Calculus: A Model of Service-Oriented Computation." In S. Drossopoulou, editor, "ESOP," volume 4960 of *Lecture Notes in Computer Science*, pages 269–283. Springer.

Web Services Business Process Working Group (**2007**). "Business Process Execution Language." `www.oasis-open.org/committees/wsbpel/`.

Web Services Choreography Working Group (**2005**). "Choreography Description Language." `http://www.w3.org/2002/ws/chor/`.

Wegiel, M. and Krintz, C. (**2008**). "XMem: type-safe, transparent, shared memory for cross-runtime communication and coordination." In R. Gupta and S. P. Amarasinghe, editors, "PLDI," pages 327–338. ACM.

Xi, H. and Pfenning, F. (**1999**). "Dependent types in practical programming." In "Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages," POPL '99, pages 214–227. ACM, New York, NY, USA.

Yoshida, N. (**1996**). "Graph Types for Monadic Mobile Processes." In V. Chandru and V. Vinay, editors, "FSTTCS," volume 1180 of *Lecture Notes in Computer Science*, pages 371–386. Springer.

Yoshida, N., Deniélou, P.-M., Bejleri, A., and Hu, R. (**2010**). "Parameterised Multiparty Session Types." In C.-H. L. Ong, editor, "FOSSACS," volume 6014 of *Lecture Notes in Computer Science*, pages 128–145. Springer.

Yoshida, N. and Vasconcelos, V. T. (**2007**). "Language Primitives and Type Discipline for Structured Communication-Based Programming Revisited: Two Systems for Higher-Order Session Communication." In "SecRet," volume 171, pages 73–93.