Programming Languages and Tools with Multiparty Session Types

Adriana Laura Voinea

Submitted in fulfilment of the requirements for the
Degree of Doctor of Philosophy

School of Computing Science
College of Science and Engineering
University of Glasgow

January 2023

Abstract

Distributed software systems are used in a wide variety of applications, including health care, telecommunications, finance, and entertainment. These systems typically consist of multiple software components, each with its own local memory, that are deployed across networks of hosts and communicate by passing messages in order to achieve a common goal. Distributed systems offer several benefits, including *scalability* — since computation happens independently on each component, it is easy and generally inexpensive to add additional components and functionality as necessary; *reliability* — since systems can be made up of hundreds of components working together, there is little disruption if a single component fails; *performance* — since work loads can be broken up and sent to multiple components, distributed systems tend to be very efficient. However, they can also be difficult to implement and analyze due to the need for heterogeneous software components to communicate and synchronize correctly and the potential for hardware or software failures.

Distributed and concurrent programming is challenging due to the complexity of coordinating the communication and interactions between the various components of a system that may be running on different machines or different threads. Behavioural types can help to address some of these difficulties by providing a way to formally specify the communication between components of a distributed system. This specification can then be used to verify the correctness of the communication between these components using static typechecking, dynamic monitoring, or a combination of the two. Perhaps the most well-known form of behavioural types are session types. They define the sequences of messages that are exchanged between two or more parties in a communication protocol, as well as the order in which these messages are exchanged. More generally, behavioural types include typestate systems, which specify the state-dependent availability of operations, choreographies, which specify collective communication behaviour, and behavioural contracts that specify the expected behaviour of a system. By using behavioural types, it is possible to ensure that the communication between components of a distributed system is well-defined and follows a set of predefined rules, which can help to prevent errors and ensure that the system behaves correctly.

The focus of this thesis is on using session type systems to provide static guarantees about the runtime behaviour of concurrent programs. We investigate two strands of work in this context. The first strand focuses on the relationship between session types and linearity. Linearity is a

property of certain resources, in this case communication channels, that can only be used once. For instance a linear variable can only be assigned once, after which it cannot be changed. This property is useful for session types because it helps to prevent race conditions and guarantees that no messages are lost or duplicated. We look at relaxing the standard access control in multiparty session types systems. This is typically based on *linear* or *affine* types, that offer strong guarantees of communication safety and session. However, these exclude many naturally occurring scenarios that make use of shared channels or need to store channels in shared data structures. We introduce a new and more flexible session type system, which allows channel references to be shared and stored in persistent data structures. We prove that the resulting language satisfies type safety, and we illustrate our type system through examples.

The second strand of research in this thesis looks at the expressive power of session types, and their connection to typestate for safe distributed programming in the Java language. Typestates are a way of annotating objects with a set of operations that are valid to perform on them at a given state. We expand the expressive power of two existing tools, use them to represent real-world case studies, and end by considering language usability and human factors.

Contents

List of Figures

Acknowledgements

I wish to express my sincere appreciation to my supervisor, Professor Simon Gay for his support, guidance, and patience throughout my time as a PhD student. He has truly gone above and beyond as a supervisor, and without his persistent help, the goal of this project would not have been realized. I am also very grateful to Ornela Dardha, my second supervisor, for her help, support and advice. It has been a pleasure to work with her. A big thank you to Prof Wim Vanderbauwhede for many interesting conversations and great advice.

I am indebted to the ABCD group for many interesting meetings and stimulating discussions, that have contributed to my understanding of programming languages.

My time as a PhD student has been greatly enriched by being a member of the FATA and PLUG groups: I have met many wonderful people and learnt many new topics through their meetings.

I would like to thank my fellow doctoral students for all of the discussions and cameraderie. In particular, I am grateful to David Maxwell for his advice and viva tips, and Oseghale Igene for his support through the write-up process.

I wish to acknowledge my family and friends for their support and encouragement throughtout this PhD. They kept me going on and this work would not have been possible without their input. My parents, Marinela and George Voinea raised me with a love of science and supported me in all my pursuits, for which I am grateful. Cristina Mihailescu has always opened her home to me and has always been helpful in numerous ways. A big thank you goes to Cristian Urlea, for his continuous encouragement, putting up with my rants while writing up, and his invaluable assistance in keeping me well caffeinated.

My studies were supported by an EPSRC PhD studentship. COST Action IC1201 (BETTY) supported my attendance at the BETTY Summer Schools in 2016. UK EPSRC grant EP/K034413/1, "From Data Types to Session Types: A Basis for Concurrency and Distribution (ABCD)" supported my attendance at the Oregon summer school in 2017. EU HORIZON 2020 MSCA RISE project 778233 "BehAPI: Behavioural Application Program Interfaces" supported my attendance at the BETTY Summer Schools in 2019.

Declaration

With the exception of chapters 1, 2 and 3, which contain introductory material, all work in this thesis was carried out by the author unless otherwise explicitly stated.

Chapter 1

Introduction

Distributed systems are ubiquitous and *communication* is an important feature and reason for their success. Communication-centric programming has proven to be one of the most successful attempts to replace shared memory for building concurrent, distributed systems. Communication is often easier to reason about and scales well as opposed to shared memory, because it allows processes to interact without the need to coordinate access to a shared state. When working with shared memory, multiple processes communicate with each other via access to shared variables or data structures. Coordinating access to shared memory can be difficult to do correctly, requiring sublte design and implemention of synchronization mechanisms to ensure thread-safety. In contrast, communication between processes involves sending messages over a communication channel, such as a network socket or message queue, rather than coordinating access to shared memory. This can be easier to understand and more scalable, as each process can operate independently and communicate with each other as needed without having to synchronize access to shared state. This makes communication a more suitable approach for systems where scalability is a must, as in the case of multi-core programming, service-oriented applications or cloud computing.

Communication between processes is typically standardized using *protocols* that specify the allowed interactions between the parties in a specific order. *Behavioural types* [80] allow these protocols to be formally specified, capturing the causal and branching structures of the communication among the participants. These type systems can help to prevent errors such as *race conditions*, where concurrent processes compete for shared resources and may produce inconsistent states or unexpected behaviors; *communication errors*, where participants have different expectations; or *message not understood errors*, where a remote participant is unable to process a message it receives.

Session types is a significant area within behavioral types, allowing the communication structures to be defined as type definitions in programming languages. These types can be used by compilers, development environments, and runtime systems for compile-time analysis or runtime monitoring. Other examples of behavioral types include typestate systems, that specify

the availability of operations or methods in an object or data structure based on the current state, choreographies, which specify collective communication behavior of a system, and behavioral contracts, which specify the expected behavior of a process or group of processes in the form of a set of rules or obligations that they must follow.

The expressiveness of session types has enabled their application in diverse contexts, targeting different programming models such as functional [134] or object-oriented programming [48], and also addressing lower-levels of application such as operating system design [52] or middleware communication protocols [132], to name just a few.

Session-typed models are used to describe open-ended systems where loosely coupled parties can synchronize to start a session on a specific public service, and then communicate privately over the session channel. Because session participants must follow their session types, their behavior is more predictable and disciplined than untyped processes. This allows session-typed models to provide strong guarantees of communication safety, protocol fidelity, and progress. *Communication safety* means that the types of the messages sent correspond to that of the expected messages; *protocol fidelity* that the interactions that occur follow the prescribed protocol; and *progress* that every message sent is eventually received, and every process waiting for a message eventually receives one. However, these strong guarantees can make it difficult to use session-typed models in scenarios that require shared resources, such as shared databases or output devices, or in implementations that use shared resources for performance reasons. To make session-typed models more practical in these situations, we propose using capability-based resource sharing, where channels are split into a reference for the channel and a capability for using the channel. This allows for more flexible resource sharing while still preserving the guarantees of session fidelity.

1.1 Research Questions

Q1 *What is the relationship between session types and linearity or affinity, and how can we check resource sharing and aliasing to guarantee type safety?*

In the context of session types, linearity, respectively affinity, is used specify the ownership of communication channels or resources. Each communication channel is then associated with a single participant, and can only be used by that participant. This is useful in providing strong guarantees for communication safety, protocol fidelity, and progress, but the use of linearity or affinity can limit their expressivity. For example in the use of session-typed models to describe scenarios that require shared resources, such as shared databases or output devices, or in implementations that use shared resources for performance reasons. In order to make session-typed models more practical and widely applicable, we look at relaxing the use of linearity or affinity for multiparty session types, while still preserving the strong guarantees provided by this model. This allows session types to be used to model

a wider range of programming scenarios and improve their expressivity.

Q2 *How can session types be adapted to support real-world case studies, and can we assess if they are really beneficial?*

Distributed systems pose many more challenges than other forms of computing, such as latency, scaling, non-determinism, independent failures, and the complexity of algorithms such as Paxos. To support realistic protocols, session type languages and tools should be able to detect the most common and critical errors. What are the ways in which session type constructs need to be adapted to support real-world case studies? How can we assess if the restrictions imposed by the type system provide sufficient value to be justifiable.

Distributed systems present many challenges, including latency, scaling, non-determinism, independent failures, and the complexity of algorithms like Paxos. In order to support realistic protocols in these systems, session type languages and tools need to be able to detect and prevent common and critical errors. We look at ways in which session type constructs need to be adapted to support real-world case studies. We also look at how we can assess if the restrictions imposed by the type system provide sufficient value to be justifiable.

1.2 Contributions

This thesis makes contributions in two strands of work: resource sharing for multiparty session π-calculus, and static type-checking of communication protocols in Java.

The contributions of Part II are:

(i) **MPST with capabilities**: we present a new version of multiparty session theory that does not use linear typing for channels, but instead uses linearly-typed capabilities. In Section 4.2 we define a multiparty session π-calculus with capabilities, and its operational semantics, and in Section 4.3 we define a MPST system for it.

(ii) **Type Safety**: in Section 4.5 we state the type safety property, and outline its proof. This property ensures that the system behaves correctly and prevents certain kinds of errors.

(iii) **Producer-Consumer Case Study**: in Chapter 5 we present case study of a producer-consumer scenario, which illustrates the use of capabilities and resource sharing in the MPST with capabilities. This case study provides a detailed account of how the system can be used to model and verify the behavior of distributed systems.

The contributions of Part III are:

(i) **HTTP Case Study**: in Section 8.2 we present a statically typechecked HTTP client which illustrates the [St]Mungo toolchain

(ii) **FTP Case Study**: in Section 8.3 we present a statically typechecked FTP client which illustrates the [St]Mungo toolchain

(iii) **Paxos Case Study**: in Section 8.4 we present a statically typechecked Paxos protocol implementation which illustrates the [St]Mungo toolchain

1.2.1 Publications

1. A. Laura Voinea, Ornela Dardha, and Simon J. Gay. Typechecking Java Protocols with [St]Mungo. In *FORTE*, volume 12136 of *Lecture Notes in Computer Science*, pages 208–224. Springer, 2020

   Primary author, based on extensions to the [St]Mungo toolchain and case study.

2. A. Laura Voinea, Ornela Dardha, and Simon J. Gay. Resource Sharing via Capability-Based Multiparty Session Types. In *IFM*, volume 11918 of *Lecture Notes in Computer Science*, pages 437–455. Springer, 2019

   Primary author.

3. Ornela Dardha, Simon J Gay, Dimitrios Kouzapas, Roly Perera, A Laura Voinea, and Florian Weber. Mungo and StMungo: Tools for Typechecking Protocols in Java. In Simon J Gay and António Ravara, editors, *Behavioural Types: from Theory to Tools*, chapter 14, pages 309–328. River Publishers, 2017

   Contributed extensions to the [St]Mungo toolchain.

4. Dimitrios Kouzapas, Ramunas Forsberg Gutkovas, A Laura Voinea, and Simon J Gay. A Session Type System for Asynchronous Unreliable Broadcast Communication. *arXiv preprint arXiv:1902.01353*, 2019

   Contributed the Paxos case study.

5. A Laura Voinea and Simon J Gay. Benefits of session types for software development. In *Proceedings of the 7th International Workshop on Evaluation and Usability of Programming Languages and Tools*, pages 26–29. ACM, 2016

   Primary author.

1.3 Thesis Outline

The remainder of this thesis is structured as follows.

Part I introduces the relevant background material. Chapter 2 introduces binary session types, surveys the literature and introduces session π-calculus, and the theoretical results and properties

that the type system satisfies. Chapter 3 introduces multiparty session types, surveys the literature and introduces multiparty session π-calculus, and the theoretical results and properties that the type system satisfies.

Part II introduces capability-based resource sharing for multiparty session types and is based on [136]. Chapter 4 introduces the calculus, the type system, the theoretical results and their detailed proofs. Chapter 5 illustrates the language on a producer-consumer example. Chapter 6 concludes and discusses related work.

Part III introduces the [St]Mungo toolchain. Chapter 7 gives an overview of the toolchain, its use and its implementation and is based on [38, 137]. Chapter 8 presents a collection of case studies that showcase and test the applicability of the toolchain to typecheck protocols, and is based on [87, 137].

Part IV concludes. Chapter 9 reprises the contributions and discusses directions for future work.

Part I

Background

In this first part we present background material required for the remainder of the thesis, and survey the literature on session types. We begin by introducing binary session types, a behavioural type system for formally specifying communication protocols between two participants. Session types capture the sequence of messages exchanged and the types of individual messages. This is encoded as types for channel endpoints, and enables conformance to protocols to be verified by type checking. We look at various approaches with particular focus on the $\pi - calculus$. Next, we look at implementations of binary session types in programming languages.

The second half of the background describes multiparty session types, which generalise binary session types by supporting more sophisticated protocols between any number of participants. They give a bird's-eye view of the interactions between multiple participants as a global type, which can then be projected into local session types. We look at various approaches with a focus on the multiparty session $\pi - calculus$ system. Next, we look at implementations of multiparty session types in programming languages.

Chapter 2

Session types

Session types, first introduced by Honda *et al.* [68, 71], are types for protocols, and describe both the type and the order of messages. In this section we are concerned with binary session types, which describe communication between exactly two participants. A session type describes the communication pattern from the point of view of one of the participants.

We illustrate the concepts of session types using a classic example from the literature, namely the math server example from Gay and Hole [61]. We show the types for this in Figure 2.1. The server offers a *choice* (branch) between two services, addition of two integers (represented by `plus`) and equality test between two integers (represented by `eq`). The `plus` option *receives* as input two integers —`?Int.?Int.`, and *sends* the result of their addition back to the client before terminating— `!Int.end`. The `eq` option *receives* two integers — `?Int.?Int.`, and *sends* the result of the equality test to the client before terminating — `!Bool.end`.
A key notion in session types is **duality**. Duality captures the idea that endpoints of the same session should be used in complementary ways: when a session type specifies the output of a message, its dual specifies the input of a message with the same type; when a session type specifies an internal choice (selection), the dual specifies an external choice (branching). The type for the math server's Client Figure 2.1 is the dual of the Server type. Whenever the Server offers a choice (&) the client makes a selection ($\oplus$), and vice versa. Whenever the Server receives a value, the Client sends a value, and vice versa. Duality ensures that the behaviour of the server matches the behaviour of the client and hence they are able to communicate with each other correctly. We give the formal definition of duality later on in Figure 2.6.

Session delegation is another important feature related to session types. Session delegation allows a process to transfer its role in the communication to another process, allowing the second

$$\mathsf{Server} = \& \begin{Bmatrix} \texttt{plus} : \texttt{?Int.?Int.!Int.end} \\ \texttt{eq} : \texttt{?Int.?Int.!Bool.end} \end{Bmatrix} \quad \mathsf{Client} = \oplus \begin{Bmatrix} \texttt{plus} : \texttt{!Int.!Int.?Int.end} \\ \texttt{eq} : \texttt{!Int.!Int.?Bool.end} \end{Bmatrix}$$

Figure 2.1: Math Server Types

process to take over the communication for the rest of the session. This is typically done by sending a message that contains a reference to the session endpoint, which represents the state of the session, to the second participant. This second participant can use the session endpoint to send and receive messages according to the session type specification, just as the initial process would have been able to do. Session delegation can be useful for a number of different contexts. For instance when different parts of a session need to be fulfilled by different processes such as a bank transfer where a process implements authentication and another process implements the transfer itself. Another example is when the a process is no longer able to continue communicating due to an error or other failure and hands off the session to a process that can take over.

Session types and linearity. A source of inspiration for session types has been linear logic [65]. In order to maintain session fidelity, i.e. the session channel has the expected structure, and ensure that all communication actions in a session type occur, session type systems require each channel endpoint to be used by exactly one participant at a time. To ensure this channel endpoints were originally treated as linear resources. This approach is reinforced by several connections between session types and other linear type theories: the encodings of binary session types and multiparty session types into linear π-calculus types [39, 122]; the Curry-Howard correspondence between binary session types and linear logic [20, 139]; the connection between multiparty session types and linear logic [23, 24]. Some session type systems generalise linearity. Vasconcelos [133] allows a session type to become non-linear, and shareable, when it reaches a state that is invariant with every subsequent message. Mostrous and Vasconcelos [95] define *affine* session types, in which each endpoint must be used at most once and can be discarded with an explicit operator. In Fowler *et al.*'s [56] implementation of session types for the Links web programming language, affine typing allows sessions to be cancelled when exceptions (including dropped connections) occur. Caires and Pérez [19] use monadic types to describe cancellation (i.e. affine sessions) and non-determinism. Pruiksma and Pfenning [120] use adjoint logic to describe session cancellation and other behaviours including multicast and replication.

Session types were initially developed in the context of the π-calculus [68, 71]. Since then, the concept has been incorporated into different areas including functional and object-oriented languages.

We now describe the classic session π-calculus, its syntax, semantics, and typing system. Our formulation is based on Gay and Hole [61], but uses the popular double binder restriction introduced by Vasconcelos [135]. We present the type safety result in this setting.

2.1 Syntax and Semantics

The syntax of processes is defined by the grammar in Figure 2.2.

The **inaction 0** represents a terminated process. The **parallel composition** $P \mid Q$ represents two processes that can execute concurrently, and potentially communicate. The **session restriction**

$$
\begin{array}{lll}
P,Q & ::= \mathbf{0} \mid P \mid Q \mid !P & \text{inaction, composition, replication} \\
& \mid \quad x!\langle v\rangle.P \mid x?(y).P & \text{send, receive} \\
& \mid \quad x \triangleleft \ell_j.P \mid x \triangleright \{\ell_i : P_i\}_{i\in I} & \text{selection, branching} \\
& \mid \quad (\nu\, xy)P \mid (\nu\, x)P & \text{session restriction, channel restriction} \\
v & ::= x \mid \mathsf{true} \mid \mathsf{false} \mid 0 \mid 1\ldots & \text{variable, base value}
\end{array}
$$

Figure 2.2: Syntax of π-calculus with session types

$(\nu\, xy)P$ declares a new session with two channel endpoints (co-variables) x, y, which have scope limited to process P. **Channel restriction**, $(\nu\, x)P$ creates a new standard π-calculus channel x. Process $x!\langle v\rangle.P$ **sends** a value v over channel x, and continues as P. Dually, process $x?(y).P$ uses channel x to **receive** a value used to substitute the bound variable y, then proceeds as P. Process $x \triangleleft \ell_j.P$ **selects** one of the options ℓ_j offered by a branching process $y \triangleright \{\ell_i : P_i\}_{i\in I}$ where x and y are dual channel endpoints. The process handles a selection at label ℓ_j by executing process P_j, if $j \in I$. In both branching and selection, the labels l_i ($i \in I$) are all different and their order is irrelevant. The **replication** process, $!P$, is the standard π-calculus process, and models recursive behaviour.

Values v can be either variables or base values. Variables are bound in inputs and channels are bound in restrictions. The derived notions of bound and free identifiers, alpha equivalence, and substitution are standard. $bv(P)$ denotes the set of bound variables of process P; $fv(P)$ denotes the set of free variables of process P.

The processes that implement the protocol described by the types in Figure 2.1 are as follows:

$$
\begin{array}{rcl}
\mathit{serverbody}(y) & = & y \triangleright \left\{ \begin{array}{l} \mathtt{plus} : y?(\mathit{num}_1).y?(\mathit{num}_2).y!\langle \mathit{num}_1 + \mathit{num}_2\rangle.\mathbf{0}, \\ \mathtt{eq} : y?(\mathit{num}_1).y?(\mathit{num}_2).y!\langle \mathit{num}_1 == \mathit{num}_2\rangle.\mathbf{0} \end{array} \right\} \\
\\
\mathit{clientbody}(x) & = & x \triangleleft \{\mathtt{plus} : x!\langle \mathit{num}_1\rangle.x!\langle \mathit{num}_2\rangle.x?(\mathit{result}).\mathbf{0}\}
\end{array}
$$

Before presenting the operational semantics, we introduce the notion of **structural congruence**, a relation used to write processes in some canonical form, allowing a more concise presentation of the reduction relation. Structural congruence is the smallest congruence relation on processes that satisfies the axioms in Figure 2.3, and is denoted by $\equiv$. The first three axioms say that parallel composition has the terminated process $\mathbf{0}$ as the neutral element: $P \mid \mathbf{0} \equiv P$; is commutative: $P \mid Q \equiv Q \mid P$; and is associative: $(P \mid Q) \mid R \equiv P \mid (Q \mid R)$. The next three axioms state that one can safely add or remove a session restriction to the terminated process: $(\nu\, xy)\mathbf{0} \equiv \mathbf{0}$; the order is not important: $(\nu\, xy)(\nu\, zw)P \equiv (\nu\, zw)(\nu\, xy)P$; and that one can extend the scope of the restriction to another process in parallel (called scope extrusion) if x and y are not free variables in the other process: $(\nu\, xy)P \mid Q \equiv (\nu\, xy)(P \mid Q)$. The last three axioms are similar to the previous, but for channel restriction. The axioms state that one can safely add or remove a channel restriction to the terminated process: $(\nu\, x)\mathbf{0} \equiv \mathbf{0}$, the order of channel restrictions is

$$P \mid \mathbf{0} \equiv P \tag{2.1}$$
$$P \mid Q \equiv Q \mid P \tag{2.2}$$
$$(P \mid Q) \mid R \equiv P \mid (Q \mid R) \tag{2.3}$$

$$(\nu\, xy)\mathbf{0} \equiv \mathbf{0} \tag{2.4}$$
$$(\nu\, xy)(\nu\, zw)P \equiv (\nu\, zw)(\nu\, xy)P \tag{2.5}$$
$$(\nu\, xy)P \mid Q \equiv (\nu\, xy)(P \mid Q),\ x,y \notin fv(Q) \tag{2.6}$$

$$(\nu\, x)\mathbf{0} \equiv \mathbf{0} \tag{2.7}$$
$$(\nu\, x)(\nu\, y)P \equiv (\nu\, y)(\nu\, x)P \tag{2.8}$$
$$(\nu\, x)P \mid Q \equiv (\nu\, x)(P \mid Q),\ x \notin fv(Q) \tag{2.9}$$

Figure 2.3: Structural congruence for the session π-calculus

[RSTNDCOM] $x!\langle v\rangle.P \mid x?(z).Q \longrightarrow P \mid Q[v/z]$

[RCOM] $(\nu\, xy)(x!\langle v\rangle.P \mid y?(z).Q) \longrightarrow (\nu\, xy)(P \mid Q[v/z])$

[RSEL] $(\nu\, xy)(x \triangleleft \ell_j.P \mid y \triangleright \{\ell_i : P_i\}_{i\in I}) \longrightarrow (\nu\, xy)(P \mid P_j)\, j \in I$

[RSTNDRES] $\dfrac{P \longrightarrow Q}{(\nu\, v)P \longrightarrow (\nu\, v)Q}$ [RRES] $\dfrac{P \longrightarrow Q}{(\nu\, vw)P \longrightarrow (\nu\, vw)Q}$

[RPAR] $\dfrac{P \longrightarrow P'}{P \mid Q \longrightarrow P' \mid Q}$ [RCONG] $\dfrac{P \equiv P' \qquad P' \longrightarrow Q' \qquad Q \equiv Q'}{P \longrightarrow Q}$

Figure 2.4: Semantics of the session π-calculus

not important: $(\nu\, x)(\nu\, y)P \equiv (\nu\, y)(\nu\, x)P$, and that one can extend the scope of the restriction to another process in parallel if x is not a free variable in the other process: $(\nu\, x)P \mid Q \equiv (\nu\, x)(P \mid Q)$.

The operational semantics is given in terms of the **reduction relation** defined by the rules in Figure 2.4. Rule [RSTNDCOM] models standard π-calculus communication: the sending process on the left of $\mid$ sends a value v on channel x, while the receiving process on the right receives the value on the same channel and substitutes the placeholder z with it. Rule [RCOM] models session communication: the sending process on the left of $\mid$sends value v on channel endpoint x, while the receiving process on the right receives the value on endpoint y and substitutes the placeholder z with it. Rule [RSEL] models choice: the selecting process on the left communicates its choice ℓ_j and continues as P; while the branching process on the right receives the label and continues as the corresponding P_j. In both [RCOM] and [RSEL] the communicating processes have prefixes that are co-variables according to the restriction $(\nu\, xy)$. Rules [RRES], [RSTNDRES], and [RPAR] state that communication can happen under session restriction, channel restriction, and

parallel composition. The structural rule [RCONG] states that reduction is closed under structural congruence.

Going back to our example, the complete client-server system is structured as follows. The client creates a session channel with endpoints x : Client and y : Server and sends one end of it to the server along a standard channel a of type #Server. The system is defined as follows:

$$server = a?(y).serverbody(y)$$

$$client = a!\langle y\rangle.(\nu\, xy)clientbody(x)$$

The system reduces by communication on a ([RSTNDCOM]) and scope extrusion (the structural congruence axiom in eq. (2.6) that allows the scope of session restriction to be extended to another process in parallel), resulting in a private connection between client and server.

$$(\nu\, xy)(clientbody(x)\,|\,serverbody(y))$$

Assuming that the $clientbody(x)$ process chooses `plus` the system then reduces as follows:

$$\begin{array}{ll}
\text{[RSEL]} & (\nu\, xy)(clientbody(x) \mid serverbody(y)) \longrightarrow \\
\text{[RCOM]} & (\nu\, xy)(x!\langle num_1\rangle.x!\langle num_2\rangle.x?(result).\mathbf{0} \mid y?(num_1).y?(num_2).y!\langle num_1+num_2\rangle.\mathbf{0}) \longrightarrow \\
\text{[RCOM]} & (\nu\, xy)(x!\langle num_2\rangle.x?(result).\mathbf{0} \mid y?(num_2).y!\langle num_1+num_2\rangle.\mathbf{0}) \longrightarrow \\
\text{[RCOM]} & (\nu\, xy)(x?(result).\mathbf{0} \mid y!\langle num_1+num_2\rangle.\mathbf{0}) \longrightarrow \\
 & (\nu\, xy)(\mathbf{0} \mid \mathbf{0})
\end{array}$$

2.2 Types and Subtypes

Session Types, given in Figure 2.5, describe the expected usage of a channel. The syntax of types is given by two syntactic categories: one for session types, S, and the other for standard π-calculus types, T, in which we also include session types.

Looking at session types we have: $!T.S$ and $?T.S$ which represent channel types for sending and receiving a value of type T before proceeding according to type S. Selection $\oplus\{\ell_i : S_i\}_{i\in I}$ and branching $\&\{\ell_i : S_i\}_{i\in I}$ are sets of distinctly labelled session types which indicate internal choice (only one of the labels will be chosen) and external choice (labels being offered for selection). Type `end` represents a terminated session, types `t` and μ`t`.S model recursion. Types T include standard channel types, session types, ground types representing booleans and integers, and recursive types. As in [133] we use `un(T)` and `lin(T)` qualifiers to distinguish between unrestricted types which correspond to standard channel types and linear types which correspond to session types.

Predicates `un(T)` and `lin(T)` (whether a type T is unrestricted or linear) are defined as follows:

- `un(T)` if and only if $T =$ `end`, or $T =$ `B`, or $T = \#T$

Session types	S	::=	$!T.S$	send
		|	$?T.S$	receive
		|	$\oplus\{\ell_i : S_i\}_{i\in I}$	selection
		|	$\&\{\ell_i : S_i\}_{i\in I}$	branching
		|	$\mathtt{end}$	end
		|	$\mathtt{t}$	type variable
		|	$\mu\mathtt{t}.S$	recursive type
Ground types	B	::=	$\mathtt{Int} \mid \mathtt{Bool}$	
Types	T	::=	$\#T$	channel type
		|	S	session type
		|	$\mathtt{t}$	type variable
		|	$\mu\mathtt{t}.T$	recursive type
		|	B	ground type

Figure 2.5: Syntax of session types

- $\mathtt{lin}(\mathtt{T})$ if and only if $T = !S.U$, or $T = \oplus\{\ell_i : S_i\}_{i\in I}$, or $T = \&\{\ell_i : S_i\}_{i\in I}$, or $T = ?S.U$.

$$\overline{\mathtt{end}} \triangleq \mathtt{end} \qquad \overline{\mathtt{B}} \triangleq \mathtt{B}$$

$$\overline{!S.T} \triangleq ?S.\overline{T} \qquad \overline{?S.T} \triangleq !S.\overline{T}$$

$$\overline{\oplus\{\ell_i : S_i\}_{i\in I}} \triangleq \&\{\ell_i : \overline{S}_i\}_{i\in I} \qquad \overline{\&\{\ell_i : S_i\}_{i\in I}} \triangleq \oplus\{\ell_i : \overline{S}_i\}_{i\in I}$$

$$\overline{\mathtt{t}} \triangleq \mathtt{t} \qquad \overline{\mu\mathtt{t}.S} \triangleq \mu\mathtt{t}.S$$

Figure 2.6: Type duality for session types

Type duality, defined in Figure 2.6 is standard, as in seminal works [71, 133]. Duality is defined on session types only, undefined otherwise. The dual of the terminated channel type is itself. The dual of a receive type is a send type and vice versa, and the dual of a branching type is a selection type and vice versa. While this inductive definition of duality function is the most common used in session type literature it only accounts for tail recursive session types. With a tail recursive session type, the recursive type refers to itself at the end of the type, rather than anywhere else. Independently, Bono and Padovani [17] and Bernardi and Hennessy [9] observe that this duality definition is not adequate in the presence of non-tail-recursive types such as $\mu\mathtt{X}.!X.\mathtt{end}$. To overcome this Bernardi and Hennessy [9], and Lindley and Morris [91] introduce alternative more general definitions. In [62] Gay *et al.*survey definitions of session type duality in the presence of recursion. They compare the definitions introduced in [9] and [91], give an alternative streamlined formulation of these, and prove equivalence between them.

$$[\text{SEND}]\ \frac{}{\texttt{end} \leqslant \texttt{end}} \qquad [\text{SCHAN}]\ \frac{T \leqslant U \qquad U \leqslant T}{\#T \leqslant \#U}$$

$$[\text{SSND}]\ \frac{T \leqslant U \qquad V \leqslant W}{!T.V \leqslant !U.W} \qquad [\text{SRCV}]\ \frac{U \leqslant T \qquad V \leqslant W}{?T.V \leqslant ?U.W}$$

$$[\text{SBR}]\ \frac{i \in I \qquad S_i \leqslant S'_i}{\&\{\ell_i : S_i\}_{i\in I} \leqslant \&\{\ell_i : S'_i\}_{i\in I\cup J}} \qquad [\text{SSEL}]\ \frac{i \in I \qquad S_i \leqslant S'_i}{\oplus\{\ell_i : S_i\}_{i\in I\cup J} \leqslant \oplus\{\ell_i : S'_i\}_{i\in I}}$$

$$[\text{S}\mu\text{L}]\ \frac{S\{\mu\texttt{t}.S/\texttt{t}\} \leqslant S'}{\mu\texttt{t}.S \leqslant S'} \qquad [\text{S}\mu\text{R}]\ \frac{S \leqslant S'\{\mu\texttt{t}.S'/\texttt{t}\}}{S \leqslant \mu\texttt{t}.S'}$$

Figure 2.7: Subtyping for session types.

Subtyping. Subtyping is a preorder relationship on types that allows the value of a subtype to be used in place of the value of its supertype. If U is a subtype of T then a value of type U is also a value of type T, and a variable of type U can always replace a variable of type T. The operations available on values of type T are also available on values of type S.

We give the definition of subtyping for session types in the style of Gay and Hole [61] in Figure 2.7. Note that the double line in the rules indicates that they should be interpreted coinductively [116, § 21]. A subtyping judgement of the form $U \leqslant T$ asserts that U is a subtype of T and T is a supertype of U. A type construct is said to be covariant in an argument if it preserves the direction of subtyping in that argument, contravariant if it inverses the direction of subtyping, and invariant if it is both covariant and contravariant in that argument (allowing only type equivalence). [SEND] relates terminated channel types. Rule [SCHAN] specifies invariance in the message type. Rule [SSND] specifies covariance in the message type and in the continuation type. Rule [SRCV] specifies contravariance in the message type and covariance in the continuation type. Rules [SBR] and [SSEL] specify that branching is covariant, and selection contravariant, in the set of labels. Intuitively, the *subtyping relation* says that a session type S is "smaller" than S' when S is "less demanding" than S' i.e., when S allows more internal choices, and imposes fewer external choices, than S'. More internal choices are allowed because a subtype adds additional behavior to the behavior defined by its supertype. Which means that a subtype can be more specific and can provide more specialized behavior than its supertype. For example Server_1 below provides an additional behaviour for negative numbers. At the same time, subtyping imposes fewer external choices because it allows a value of a subtype to be used in any context where a value of its supertype is expected. [SμL] and [SμR] relate types up-to their unfolding.

For example an implementation that is written to work with a supertype will also work with any of its subtypes, without having to be modified. For example a process implementing the type Server_1 below can be used in place of a process implementing Server. Server_1 is a subtype for

the math server:

$$\mathsf{Server}_1 = \& \left\{ \begin{array}{l} \texttt{plus} : \texttt{?Int.?Int.!Int.end} \\ \texttt{eq} : \texttt{?Int.?Int.!Bool.end} \\ \texttt{neg} : \texttt{?Int.!Int.end} \end{array} \right\}$$

Since $\mathsf{Server} \leqslant \mathsf{Server}_1$ it is safe for the new server, a process that also implements the `neg` service, to communicate on channel y of type Server: this just means that the `neg` service is never used.

Subtyping for session types was first introduced by Gay and Hole [61], and has been used in other works by various authors [25–27, 59]. Padovani [108, 109, 111] has considered another form of subtyping, called fair subtyping. Their approaches correspond to safe substitutability of channels, rather than processes. A subtyping relation with the opposite direction has been used by Honda, Yoshida, Mostrous and other authors [22, 43, 49, 94, 96]. Chen *et al.* [30–32] have studied the preciseness of subtyping: the subtyping relation is sound and complete for safe substitutability. Their approaches correspond to safe substitutability of processes. In [60], Gay presents a thorough comparison the two definitions and shows how they can be unified. Here we use the "channel oriented" order of Gay and Hole [61].

2.3 Typing

Definition 2.3.1 (Typing Contexts)**.** Γ denotes a partial mapping from variables to types:

$$\Gamma \ ::= \ \varnothing \ \mid \ \Gamma, x : T$$

Addition of a typed name to an environment, $\Gamma + x : T$, is defined by:

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

Typing judgements are inductively defined by the rules in Figure 2.8. Typing judgements for values have the form $\Gamma \vdash v : S$, stating that a value v has type S in the typing context Γ; and for processes have the form $\Gamma \vdash P$, stating that a process P uses channels as specified by the types in Γ.

Rule [TVAR] types variables, [TVAL] values, and [TINACT] a terminated process. [TSUB] allows process P typed under $\Gamma, x : S$ to use a channel x requiring a less demanding type S'. [TPAR] types the parallel composition of two processes, using context addition given in definition 2.3.1 to ensure that linear variables are used either in Γ_1 or in Γ_2, but never in both. [TREP] types the replication process under the assumption that context Γ is unrestricted. [TSTNDRES] states that the restriction $(\nu\, x)P$ is well typed if P is well typed and variable x is of standard channel type $\#T$. [TRES] requires the two channel endpoints x, y to have dual types. [TRCV] states that receiving

$$[\text{TVAR}]\ \frac{\mathtt{un}(\Gamma)}{\Gamma, x : S \vdash x : S} \qquad [\text{TVAL}]\ \frac{\mathtt{un}(\Gamma) \quad v \in B}{\Gamma \vdash v : B}$$

$$[\text{TINACT}]\ \frac{\mathtt{un}(\Gamma)}{\Gamma \vdash \mathbf{0}} \qquad [\text{TSUB}]\ \frac{\Gamma, x : T \vdash P \quad T \leqslant T'}{\Gamma, x : T' \vdash P}$$

$$[\text{TPAR}]\ \frac{\Gamma_1 \vdash P \quad \Gamma_2 \vdash Q}{\Gamma_1 + \Gamma_2 \vdash P \,|\, Q} \qquad [\text{TREP}]\ \frac{\mathtt{un}(\Gamma) \quad \Gamma \vdash P}{\Gamma \vdash !P}$$

$$[\text{TRES}]\ \frac{\Gamma, x : S, y : \overline{S} \vdash P}{\Gamma \vdash (\nu\, xy)P} \qquad [\text{TSTNDRES}]\ \frac{\Gamma, x : \#T \vdash P \quad \mathtt{un}(\mathtt{T})}{\Gamma \vdash (\nu\, x)P}$$

$$[\text{TRCV}]\ \frac{\Gamma, x : S, y : T \vdash P}{\Gamma, x : ?T.S \vdash x?(y).P} \qquad [\text{TSND}]\ \frac{\Gamma, x : S \vdash P}{(\Gamma, x : !T.S) + y : T \vdash x!\langle v \rangle.P}$$

$$[\text{TSEL}]\ \frac{\Gamma, x : S_j \vdash P \qquad j \in I}{\Gamma, x : \oplus\{\ell_i : S_i\}_{i \in I} \vdash x \triangleleft \ell_j.P} \qquad [\text{TBR}]\ \frac{\Gamma, x : S_i \vdash P_i \qquad i \in I}{\Gamma, x : \&\{\ell_i : S_i\}_{i \in I} \vdash x \triangleright \{\ell_i : P_i\}_{i \in I}}$$

Figure 2.8: Typing rules for the π-calculus with session types

on channel x is well typed if the type associated with it is of compatible send type and process P is well-typed with the continuation session types. [TSND] states that sending on channel x is well typed if the type associated with it is of compatible receive type and P is well-typed with the continuation session types. [TSEL] states that selection on channel x is well typed if the type associated with it is of compatible selection type and the continuations P are well-typed with the continuation session type S_j. [TBR] states that branching on channel x is well typed if the type associated with it is of compatible branching type and the continuations $P_i, \forall i \in I$ are well-typed with the continuation session types.

2.4 Main Results

In this section we present the main properties satisfied by the session type system presented. The following lemmas and theorems are proven in [61].

Weakening allows introduction of new unrestricted channels in a typing context. It holds only for unrestricted channels, for linear ones it is unsound since when a linear channel is in a typing context it is used in the process it types. The weakening lemma is useful when we need to relax the typing assumptions for a process and include new typing assumptions of variables not free in the process.

Lemma 2.4.1 (Unrestricted Weakening)**.** *If* $\Gamma \vdash P$ *and* T *is not linear then* $\Gamma, x : T \vdash P$.

As opposed to weakening, strengthening allows us to remove unrestricted channels from the typing context when they are not among the free variables of the process being typed.

Lemma 2.4.2 (Strengthening for expressions)**.** *If* $\Gamma, x : S \vdash v : U$ *and* $x \notin fv(v)$ *then* $\mathtt{un}(\mathtt{S})$ *and* $\Gamma \vdash v : U$.

Lemma 2.4.3 (Strengthening)**.** *If* $\Gamma, x : S \vdash P$ *and* $x \notin fv(P)$ *then* $\mathtt{un}(\mathtt{S})$ *and* $\Gamma \vdash P$.

Lemma 2.4.4 (Substitution)**.** *Let* $\Gamma, w : W \vdash P$ *and* $Z \leqslant W$. *If* $\Gamma + z : Z$ *is defined then* $\Gamma + z : Z \vdash P[v/z]$.

Lemma 2.4.5 (Type Preservation under $\equiv$ for session π-calculus)**.** *Let* $\Gamma \vdash P$ *and* $P \equiv P'$*, then* $\Gamma \vdash P'$.

Theorem 2.4.6 (Subject Reduction)**.** *If* $\Gamma \vdash P$ *and* $P \longrightarrow Q$ *then* $\Gamma \vdash Q$.

A redex ("reducible expression") is a process of the form: $x!\langle \tilde{v} \rangle.P \,|\, y?(\tilde{z}).Q$ or $x \triangleleft \ell_j.P \,|\, y \triangleright \{\ell_i : P_i\}_{i \in I}$ with $j \in I$.

Theorem 2.4.7 (Type safety)**.** *A process is well-formed if for each of its structural congruent processes* $(\nu\, \widetilde{xy})(P \,|\, Q)$*, the following conditions hold:*

- *if P and Q are processes prefixed at the same variable, then they are of the same nature (input, output, branch, selection), and*
- *if P is prefixed in* x_i *and Q is prefixed in* y_i *where* $x_i \in \tilde{x}$, $y_i \in \tilde{y}$ *, then* $P \,|\, Q$ *is a redex.*

The subject reduction theorem guarantees that as P reduces, each subsequent process P' is typable under the same environment, and the type safety theorem guarantees that P' has no immediate communication errors.

2.5 Implementations

Session types have inspired the design of several new programming languages. **SePi** [5, 57, 133] is a concurrent, message-passing programming language based on session π-calculus, featuring a simple form of refinement types. The language features synchronous, bi-directional channel-based communication. Primitives are used for *sending*, *receiving*, *branching*, and *selecting* messages. Channel interactions are statically verified against the session types describing the type and order of messages exchanged, as well as the number of processes that may share a channel. **SILL** [115, 131] is a functional programming language with a linear contextual monad for session-typed message-passing concurrency. It is based on the Curry-Howard isomorphism between intuitionistic linear logic and session-typed concurrency. **Concurrent C0** [142] is a type-safe programming language closely resembling C. It allows the creation of concurrent

processes and session typed communication channels. **Links** [33] is a functional programming language designed for tierless web applications (the system generating code for both front and back-end tiers) that offers native support for binary session types [56, 92].

Session types have also been implemented into many mainstream programming languages. The **Session Java (SJ)** language [79] builds on earlier work [47, 48, 50] on session types in an object-oriented setting to add binary session type channels to Java. SJ has been applied to a range of situations including scientific computation [104], and has been extended to event-driven programming [75]. Session primitives are represented as APIs, and Java syntax is extended with communication statements to allow typechecking.

Neubauer and Thiemann [97] provide the first encoding of session-typed core calculus into **Haskell** using type classes with functional dependencies. They avoid aliasing by automatically threading the implicit channel through the computation. Pucella and Tov [121] enforce linear channel usage by using a parameterised monad [4] indexed by type-level pre- and post-conditions on session environments; they allow communication along multiple channels through an explicitly-managed stack (representing session environments). Both lines of work feature first-order sessions with branch/select and recursion, but without delegation. Imai *et al.* [83] extend Pucella-Tov's approach with delegation, type inference and a more user-friendly approach to handling multiple channels, by replacing the aforementioned stack with a De Bruijn index encoding which is handled implicitly at the type level. Lindley and Morris [90] follow Polakow's [118] approach to embedding of a linear λ-calculus in Haskell. They embed GV [63, 91, 139] (a session-typed linear λ-calculus) in Haskell, allowing first-class channel endpoints. Orchard and Yoshida [107] encode session-typed π-calculus into FPCF, a parallel variant of PCF [117] with a general, parameterised effect system. They use this encoding together with an approach for embedding effect systems in Haskell [106] to provide an implementation of session-typed channels in Concurrent Haskell. All of these implementations leverage Haskell's type system to provide static linearity checks, however each comes with its own deficiencies. Recent work by Bernardy *et al.* [10] adds first-class linear types to Haskell. So, it would be interesting to see a session type library for Haskell using first-class linear types, rather than additional artefacts of embedding linearity.

Imai *et al.* [82] propose session-ocaml, a library for session-typed programming in **OCaml**. They use a parameterised monad, and polarities, which view a session type from the point of the view of a client or a server. The authors additionally make use of lenses to more cleanly manipulate the stack of session channels. Padovani [113] describes **FuSe**, a lightweight implementation of binary session types in OCaml, that verifies message ordering statically and linearity violations dynamically. Although runtime checking of linearity results in fewer static guarantees, it results in a particularly clean library design and implementation, in particular allowing first-class manipulation of channel endpoints. Padovani [112] builds upon the work of Thiemann and Vasconcelos [130] by being the first to implement context-free session types, in the setting of the FuSe. Padovani reformulates context-free sessions to make use of resumptions to implement the

sequencing required by context-free session types. To safely implement resumptions, session types are ascribed with identities. An advantage of Padovani's approach is that type equivalence is no longer needed in the type system of terms, but only to reason about the metatheory, at the cost of additional syntactic markers.

Chapter 3

Multiparty Session Types

3.1 Introduction

Multiparty session types, introduced by Honda *et al.* [72, 73] and Bettini *et al.* [11], generalise binary session types by allowing a top-down description of the interactions between multiple participants in a protocol.

The multiparty session types methodology is as follows:

- the protocol is specified as a *global type*, G, that gives a global view of the interactions between all participants, or *roles*
- the global type G is *projected* to each role to obtain a *(local) session type*
- the session types are assigned to *communication channels*, used by implementing processes, which can then be typechecked for conformance

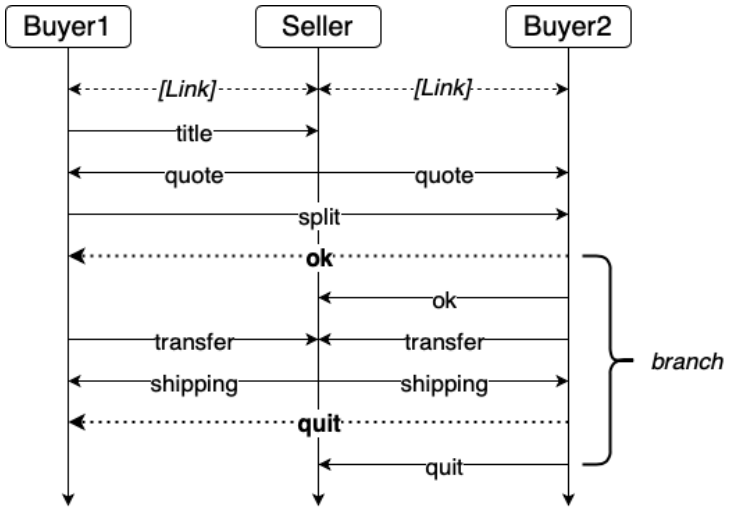

Figure 3.1: Two-buyer Protocol

$$G = \mathbf{b_1} \to \mathbf{s}\colon \mathtt{title}(\mathtt{Str}).\mathbf{s} \to \mathbf{b_1}\colon \mathtt{quote}(\mathtt{Int}).\mathbf{s} \to \mathbf{b_2}\colon \mathtt{quote}(\mathtt{Int}).\mu \mathrm{t}.\mathbf{b_1} \to \mathbf{b_2}\colon \mathtt{split}(\mathtt{Int}).$$
$$\mathbf{b_2} \to \mathbf{b_1}\colon \begin{Bmatrix} \mathtt{ok}.\mathbf{b_2} \to \mathbf{s}\colon \mathtt{ok}.\mathbf{b_1} \to \mathbf{s}\colon \mathtt{buy}(\mathtt{Int}).\mathbf{b_2} \to \mathbf{s}\colon \mathtt{buy}(\mathtt{Int}).\mathtt{end}, \\ \mathtt{no}.\mathbf{b_2} \to \mathbf{s}\colon \mathtt{no.t}, \\ \mathtt{quit}.\mathbf{b_2} \to \mathbf{s}\colon \mathtt{quit.end} \end{Bmatrix}$$

Figure 3.2: TwoBuyer Global Type

$$S_{\mathbf{b_1}} = \mathbf{s} \oplus \mathtt{title}(\mathtt{Str}).\mathbf{s} \& \mathtt{quote}(\mathtt{Int}).\mu \mathrm{t}.\mathbf{b_2} \oplus \mathtt{split}(\mathtt{Int}).\mathbf{b_2} \& \begin{Bmatrix} \mathtt{ok}.\mathbf{s} \oplus \mathtt{buy}(\mathtt{Int}).\mathtt{end} \\ \mathtt{no}.\mathbf{s} \oplus \mathtt{no.t} \\ \mathtt{quit.end} \end{Bmatrix}$$

$$S_{\mathbf{b_2}} = \mathbf{s} \& \mathtt{quote}(\mathtt{Int}).\mu \mathrm{t}.\mathbf{b_1} \& \mathtt{split}(\mathtt{Int}).\mathbf{b_1} \oplus \begin{Bmatrix} \mathtt{ok}.\mathbf{s} \oplus \mathtt{buy}(\mathtt{Int}).\mathtt{end} \\ \mathtt{no}.\mathbf{s} \oplus \mathtt{no.t} \\ \mathtt{quit}.\mathbf{s} \oplus \mathtt{quit.end} \end{Bmatrix}$$

$$S_{\mathbf{s}} = \mathbf{b_1} \& \mathtt{title}(\mathtt{Str}).\mathbf{b_1} \oplus \mathtt{quote}(\mathtt{Int}).\mathbf{b_2} \oplus \mathtt{quote}(\mathtt{Int}).$$
$$\mu \mathrm{t}.\mathbf{b_2} \& \begin{Bmatrix} \mathtt{ok}.\mathbf{b_1} \& \mathtt{buy}(\mathtt{Int}).\mathbf{b_2} \& \mathtt{buy}(\mathtt{Int}).\mathtt{end} \\ \mathtt{no.t} \\ \mathtt{quit.end} \end{Bmatrix}$$

Figure 3.3: TwoBuyer Session Types

We illustrate the concepts of multiparty session types using a canonical example from the literature, namely the two-buyer protocol, intended to be representative of financial protocols.

The two-buyer protocol describes the scenario in Figure 3.1. We show the full global type for this example in Figure 3.2.

1. Buyer1 ($\mathbf{b_1}$) requests the price of a book title from the Seller ($\mathbf{s}$): $\mathbf{b_1} \to \mathbf{s}\colon \mathtt{title}(\mathtt{Str})$
2. The Seller sends the quote to Buyer1: $\mathbf{s} \to \mathbf{b_1}\colon \mathtt{quote}(\mathtt{Int})$, and to Buyer2: $\mathbf{s} \to \mathbf{b_2}\colon \mathtt{quote}(\mathtt{Int})$
3. Buyer1 sends Buyer2 the amount that Buyer2 should pay: $\mathbf{b_1} \to \mathbf{b_2}\colon \mathtt{split}(\mathtt{Int})$
4. Buyer2 can choose to:
   - Accept the offer, at which point Buyer1 buys the item: $\mathbf{b_2} \& \mathtt{ok}.\mathbf{b_2} \to \mathbf{s}\colon \mathtt{ok}.\mathbf{b_1} \to \mathbf{s}\colon \mathtt{buy}(\mathtt{Int}).\mathbf{b_2} \to \mathbf{s}\colon \mathtt{buy}(\mathtt{Int}).\mathtt{end}$
   - Reject the offer, and await another offer from Buyer 1: $\mathbf{b_2} \& \mathtt{no}.\mathbf{b_2} \to \mathbf{s}\colon \mathtt{no.t}$
   - End the protocol: $\mathbf{b_2} \& \mathtt{quit}.\mathbf{b_2} \to \mathbf{s}\colon \mathtt{quit.end}$

The projections of G describe the local actions that programs must implement to play the roles in G. The types shown in Figure 3.3, $S_{\mathbf{b_1}}, S_{\mathbf{b_2}}, S_{\mathbf{s}}$ are session types obtained by projecting G

onto the three roles $\mathbf{b_1}, \mathbf{b_2}, \mathbf{s}$. $S_{\mathbf{b_1}}$ captures only the communication behaviour involving role $\mathbf{b_1}$: it sends ($\oplus$) a `title` to $\mathbf{s}$, receives ($\&$) a `quote` back, then asks $\mathbf{b_2}$ to `split` the price, to which $\mathbf{b_2}$ can reply with either: `ok` in which case they `buy` the item; `no` in which case $\mathbf{b_1}$ recursively retries proposing a different `split`; or `quit` in which case the order is cancelled. The types for Buyer2, $S_{\mathbf{b_2}}$, and for Seller, $S_{\mathbf{s}}$ follow the same intuition. In multiparty session type systems these types are assigned to channels, and the endpoint programs that use them are checked for correct usage, i.e. the program implementing $\mathbf{b_1}$ is checked against $S_{\mathbf{b_1}}$. Endpoint programs are in turn formalised as *processes* in the π-calculus extended with multiparty session primitives.

Typing ensures that processes behave as specified, i.e. they interact according to the global type G. This is typically achieved by imposing well-formedness conditions to global types, and consistency restrictions when processes are type-checked. Recent work by Scalas and Yoshida [125] identifies the limitations on the expressiveness of previous multiparty session typing systems and in many cases flawed subject reduction proofs, due to a conservative requirement of consistency. Scalas and Yoshida propose a more general, expressive, and simpler system based on generic type systems for the π-calculus [81]. The authors define run-time safety properties such as liveness or deadlock freedom, and show how protocol conformance can be checked via a translation into formulae which can be checked by the mCRL2 [66] model checker. The system allows more expressive protocols to be described and checked against implementations, a significant advantage over both classic multiparty session type systems and model checking on their own.

We now describe multiparty session π-calculus, its types, and typing rules. Our formulation is based on Coppo *et al.* [34] and Scalas *et al.* [122]; and includes subtyping [49]. We present the type safety result in this setting, and then present the safety invariant introduced by Scalas and Yoshida [125].

We use the following conventions:

- Derivations use single lines for inductive rules and double lines for coinductive rules.
- Recursive types $\mu \mathtt{t}.T$ are always closed, i.e. without free type variables, and guarded which ensures that they are contractive: in $\mu \mathtt{t}.T$ we have $T \neq t', \forall t'$; so $\mu \mathtt{t}_1 \ldots \mu \mathtt{t}_n.\mathtt{t}_1$ is not a type.
- We define the unfolding of a recursive types as: $\mathtt{unf}(\mu \mathtt{t}.T) = \mathtt{unf}(T\{\mu \mathtt{t}.T/\mathtt{t}\})$, and $\mathtt{unf}(T) = T$ if $T \neq \mu \mathtt{t}.T'$.
- Type equality is syntactic: $\mu \mathtt{t}.T$ is not equal to $\mathtt{unf}(\mu \mathtt{t}.T)$.
- We write $P \longrightarrow Q$ for process reductions, $\longrightarrow^*$ for the reflexive and transitive closure of $\longrightarrow$, and $P \not\longrightarrow$ if and only if there is no Q such that $P \longrightarrow Q$.
- For readability, we use **brown** for global types, **blue** for local session types, and **green** for partial session types.

$$
\begin{array}{lcll}
P & ::= & \mathbf{0} & \text{inaction} \\
 & | & P \,|\, Q & \text{parallel composition} \\
 & | & (\nu\, s)P & \text{restriction} \\
 & | & \mathtt{c}[\mathbf{p}] \oplus \langle l(v) \rangle . P & \text{select} \\
 & | & \mathtt{c}[\mathbf{p}] \& _{i \in I} \{ l_i(x_i) . P_i \} & \text{branch} \\
 & | & \textbf{def } D \textbf{ in } P & \text{recursion} \\
 & | & X \langle \tilde{x} \rangle & \text{process call} \\
 & & & \\
D & ::= & X(\tilde{x}) = P & \text{process declaration} \\
 & & & \\
\mathtt{c} & ::= & x & \text{variable} \\
 & | & \mathtt{s}[\mathbf{p}] & \text{channel with role p} \\
v & ::= & \mathtt{c} & \text{channel} \\
 & | & \mathsf{true} \mid \mathsf{false} & \text{booleans} \\
 & | & 0 \mid 1 \mid \ldots & \text{base value}
\end{array}
$$

Figure 3.4: Multiparty session π-calculus

3.2 Syntax and Semantics

The syntax of the $\pi-$calculus with multiparty session types is given in Figure 3.4. A **channel** $\mathtt{c}$ can be either a variable, x, or a **channel with a role** $\mathtt{s}[\mathbf{p}]$, i.e., a multiparty communication endpoint whose user plays role $\mathbf{p}$ in the session s. **Values** v can be variables, channels with roles, or base values. The **inaction 0** represents a terminated process. The **parallel composition** $P \,|\, Q$ represents two processes that can execute concurrently, and potentially communicate. The **session restriction** $(\nu\, s)P$ declares a new session s with scope limited to process P. Process $\mathtt{c}[\mathbf{p}] \oplus \langle l(v) \rangle . P$ performs a **selection (internal choice)** towards role $\mathbf{p}$, using the channel $\mathtt{c}$: the labelled value $l(v)$ is sent, and the execution continues as process P. Dually, process $\mathtt{c}[\mathbf{p}] \& _{i \in I} \{ l_i(x_i) . P_i \}$ uses channels $\mathtt{c}$ to wait for a **branching (external choice)** from role $\mathbf{p}$: if the labelled value $l_k(v)$ is received (for some $k \in I$), then the execution continues as P_k (with x_k holding value v). Note that for all $i \in I$, variable x_i is bound with scope P_i. In both branching and selection, the labels l_i ($i \in I$) are all different and their order is irrelevant. **Process definition def** D **in** P and **process call** $X \langle \tilde{x} \rangle$ model recursion, with D being a **process declaration** $X(\tilde{x}) = P$: the call invokes X by expanding it into P, and replacing its formal parameters with their values.

While this simplified syntax does not have dedicated input/output prefixes, they can be encoded using singleton & (with *one* branch) and $\oplus$ types.

The processes that implement the roles described by the types in Figure 3.3 are defined below. A message in session $\mathtt{s}$ from role $\mathbf{p}$ to role $\mathbf{q}$ has the prefix $\mathtt{s}[\mathbf{p}][\mathbf{q}]$, where $\mathtt{s}[\mathbf{p}]$ is represented by $\mathtt{c}$ in the grammar.

$$P \mid \mathbf{0} \equiv P \tag{3.1}$$
$$P \mid Q \equiv Q \mid P \tag{3.2}$$
$$(P \mid Q) \mid R \equiv P \mid (Q \mid R) \tag{3.3}$$

$$(\nu\, s)\mathbf{0} \equiv \mathbf{0} \tag{3.4}$$
$$(\nu\, s)(\nu\, s')P \equiv (\nu\, s')(\nu\, s)P \tag{3.5}$$
$$(\nu\, s)P \mid Q \equiv (\nu\, s)(P \mid Q) \quad \text{if } s \notin fc(Q) \tag{3.6}$$

$$\mathbf{def}\ D\ \mathbf{in}\ \mathbf{0} \equiv \mathbf{0} \tag{3.7}$$
$$\mathbf{def}\ D\ \mathbf{in}\ (\nu\, s)P \equiv (\nu\, s)(\mathbf{def}\ D\ \mathbf{in}\ P) \quad \text{if } s \notin fc(P) \tag{3.8}$$
$$\mathbf{def}\ D\ \mathbf{in}\ (P \mid Q) \equiv (\mathbf{def}\ D\ \mathbf{in}\ P) \mid Q \quad \text{if } dpv(D) \cap fpv(Q) = \varnothing \tag{3.9}$$
$$\mathbf{def}\ D\ \mathbf{in}\ \mathbf{def}\ D'\ \mathbf{in}\ P \equiv \mathbf{def}\ D'\ \mathbf{in}\ \mathbf{def}\ D\ \mathbf{in}\ P \tag{3.10}$$
$$\text{if } (dpv(D) \cup fpv(D)) \cap dpv(D') = (dpv(D') \cup fpv(D')) \cap dpv(D) = \varnothing$$

Figure 3.5: Structural congruence for the π-calculus with multiparty sessions

$$Buyer_1 = \mathtt{s}[\mathbf{b_1}][\mathtt{s}] \oplus \langle \mathtt{title}(name) \rangle . \mathtt{s}[\mathbf{b_1}][\mathtt{s}] \& \{ \mathtt{quote}(price) \} . \mathbf{def}\ Loop_1(x) =$$
$$\mathtt{x}[\mathbf{b_2}] \oplus \langle \mathtt{split}(split) \rangle . \mathtt{x}[\mathbf{b_2}] \& \begin{Bmatrix} \mathtt{ok}() . \mathtt{x}[\mathtt{s}] \oplus \langle \mathtt{buy}(share) \rangle . \mathbf{0} \\ \mathtt{no}() . Loop_1(x) \\ \mathtt{quit}() . \mathbf{0} \end{Bmatrix} \mathbf{in}\ Loop_1(\mathtt{s}[\mathbf{b_1}])$$

$$Buyer_2 = \mathtt{s}[\mathbf{b_2}][\mathtt{s}] \& \{ \mathtt{quote}(price) \} . \mathbf{def}\ Loop_2(x) = \mathtt{x}[\mathbf{b_1}] \& \{ \mathtt{split}(split) \} .$$
$$\mathtt{x}[\mathbf{b_1}] \oplus \begin{Bmatrix} \mathtt{ok}() . \mathtt{x}[\mathtt{s}] \oplus \langle \mathtt{ok}() \rangle . \mathtt{x}[\mathtt{s}] \oplus \langle \mathtt{buy}(share) \rangle . \mathbf{0} \\ \mathtt{no}() . \mathtt{x}[\mathtt{s}] \oplus \langle \mathtt{no}() \rangle . Loop_2(x) \\ \mathtt{quit}() . \mathtt{x}[\mathtt{s}] \oplus \langle \mathtt{quit}() \rangle . \mathbf{0} \end{Bmatrix} \mathbf{in}\ Loop_2(\mathtt{s}[\mathbf{b_2}])$$

$$Seller = \mathtt{s}[\mathtt{s}][\mathbf{b_1}] \& \{ \mathtt{title}(name) \} . \mathtt{s}[\mathtt{s}][\mathbf{b_1}] \oplus \langle \mathtt{quote}(price) \rangle . \mathtt{s}[\mathtt{s}][\mathbf{b_2}] \oplus \langle \mathtt{quote}(price) \rangle .$$
$$\mathbf{def}\ Loop_s(x) = \mathtt{x}[\mathbf{b_2}] \& \begin{Bmatrix} \mathtt{ok}() . \mathtt{x}[\mathbf{b_1}] \& \{ \mathtt{buy}(share) \} . \mathtt{x}[\mathbf{b_2}] \& \{ \mathtt{buy}(share) \} . \mathbf{0} \\ \mathtt{no}() . Loop_s(x) \\ \mathtt{quit}() . \mathbf{0} \end{Bmatrix} \mathbf{in}\ Loop_s(\mathtt{s}[\mathtt{s}])$$

Before presenting the operational semantics, we introduce the notion of **structural congruence** $\equiv$ which is the smallest congruence relation satisfying the axioms in Figure 3.5.

The definition uses the concepts of free channels of a process, $fc(P)$; free process variables of a process, $fpv(P)$; and defined process variables of a process declaration, $dpv(D)$. We write "$s \notin fc(P)$ " to mean that there does not exist a $\mathbf{p}$ such that $\mathtt{s}[\mathbf{p}] \in fc(P)$. We use $fv(D)$ to denote the set of free variables in D. We use $dpv(D)$ to denote the set of process variables declared in D, and $fpv(P)$ for the set of process variables which occur free in P.

The semantics of the system is given in terms of the **reduction relation** defined by the

$$[\text{RCom}]\ \frac{j \in I \text{ and } fv(v) = \varnothing}{\mathtt{s}[\mathbf{p}][\mathbf{q}]\&_{i\in I}\{l_i(x_i).P_i\} \mid \mathtt{s}[\mathbf{q}][\mathbf{p}]\oplus\langle l_j(v)\rangle.P \longrightarrow P_j\{v/x_j\} \mid P}$$

$$[\text{RCall}]\ \frac{\widetilde{x} = x_1,\ldots,x_n \qquad \widetilde{v} = v_1,\ldots,v_n \qquad fv(\widetilde{v}) = \varnothing}{\mathbf{def}\, X\langle\widetilde{x}\rangle = P \,\mathbf{in}\, (X\langle\widetilde{x}\rangle \mid Q) \longrightarrow \mathbf{def}\, X\langle\widetilde{x}\rangle = P \,\mathbf{in}\, (P\{\widetilde{v}/\widetilde{x}\} \mid Q)}$$

$$[\text{RRes}]\ \frac{P \longrightarrow Q}{(\nu\, s)P \longrightarrow (\nu\, s)Q} \qquad [\text{RPar}]\ \frac{P \longrightarrow Q}{P \mid R \longrightarrow Q \mid R}$$

$$[\text{RDef}]\ \frac{P \longrightarrow Q}{\mathbf{def}\, D \,\mathbf{in}\, P \longrightarrow \mathbf{def}\, D \,\mathbf{in}\, Q} \qquad [\text{RStruct}]\ \frac{P \equiv P' \qquad P \longrightarrow Q \qquad Q \equiv Q'}{P' \longrightarrow Q'}$$

Figure 3.6: Reduction for the π-calculus with multiparty sessions

rules in Figure 3.6. Rule [RCom] models communication: it says that the parallel composition of a branch and a select process, both operating on the same session s as roles **p** and **q**, via $\mathtt{s}[\mathbf{p}]$ and $\mathtt{s}[\mathbf{q}]$, and targeting each other (i.e., $\mathtt{s}[\mathbf{p}]$ is used to branch from **q**, and $\mathtt{s}[\mathbf{q}]$ is used to select towards **p**) reduces to the corresponding continuations, with a value substitution on the receiver side. [RCall] says that a process call $X\langle\tilde{x}\rangle$ in the scope of **def** D **in** $P\ldots$ reduces by expanding $X\langle\tilde{x}\rangle$ into P, and replacing the formal parameters, $\widetilde{x}$, with the actual ones, $\widetilde{v}$. The remaining rules are standard: reduction can happen under parallel composition, restriction and process definition. By [RStruct], reduction is closed under structural congruence.

3.3 Types and Subtypes

The general methodology of multiparty session types is that system design begins with a *global type*, which specifies all of the communication among various *roles*. Given a global type G and a role **p**, *projection* yields a *session type* or *local type* $G\upharpoonright\mathbf{p}$ that describes all of the communication involving **p**. This local type can be further projected for another role **q**, to give a *partial session type* that describes communication between **p** and **q**.

Global types are given in Figure 3.7. Type $\mathtt{p}\to\mathtt{q}$: $\{l_i(U_i).G_i\}_{i\in I}$ states that role **p** sends to role **q** one of the pairwise distinct labels l_i for $i \in I$, together with a payload U_i. If the chosen label is l_j, then the interaction proceeds as G_j. Type $\mu\mathtt{t}.G$ and type variable $\mathtt{t}$ model guarded recursion. Type $\mathtt{end}$ states the termination of a protocol. We omit the braces $\{...\}$ from interactions when I is a singleton, for example $\mathbf{a} \to \mathbf{b} : \{l_i(U_i).G_i\}_{i\in\{1\}}$ is written as $\mathbf{a} \to \mathbf{b} : l_1(U_1).G_1$.

Local (session) types are given in Figure 3.7 and describe the expected usage of a channel in a communication protocol involving two or more *roles*. Local types model structured sequences of input and output actions and specify the source and target of each interaction.
The **branching type** $\mathbf{p}\ \&_{i\in I}?l_i(U_i).S_i$ describes a channel that can receive a label l_i from role

Local session type	S	::=	$\mathtt{end}$	terminated session
		$\mid$	$\mathbf{p} \oplus_{i \in I} !l_i(U_i).S_i$	selection towards role p
		$\mid$	$\mathbf{p} \,\&_{i \in I} ?l_i(U_i).S_i$	branching from role p
		$\mid$	$\mathtt{t}$	type variable
		$\mid$	$\mu \mathtt{t}.S$	recursive type
Global type	G	::=	$\mathtt{p} \to \mathtt{q}\colon \{l_i(U_i).G_i\}_{i \in I}$	interaction
		$\mid$	$\mathtt{t}$	type variable
		$\mid$	$\mu \mathtt{t}.G$	recursive type
		$\mid$	$\mathtt{end}$	termination
Partial session type	H	::=	$\mathtt{end}$	terminated session
		$\mid$	$\oplus_{i \in I} !l_i(U_i).H_i$	selection
		$\mid$	$\&_{i \in I} ?l_i(U_i).H_i$	branching
		$\mid$	$\mathtt{t}$	type variable
		$\mid$	$\mu \mathtt{t}.H$	recursive type
Ground type	B	::=	$\mathtt{Int} \mid \mathtt{Bool}$	
Payload type	U	::=	$B \mid S$ closed	ground type, session type
Environment	Γ	::=	$\varnothing \mid \Gamma, x : U \mid \Gamma, \mathtt{s}[\mathbf{p}] : S$	
	Δ	::=	$\varnothing \mid \Delta, X : \tilde{U}$	*process names*

All branch and select types have the conditions $I \neq \emptyset$ and U_i closed.

Figure 3.7: Types and environments for the π-calculus with multiparty sessions

$\mathbf{p}$ (for some $i \in I$, chosen by $\mathbf{p}$), together with a *payload* of type U_i; then, the channel must be used as S_i. **Selection** $\mathbf{p} \oplus_{i\in I} !l_i(U_i).S_i$, describes a channel that can choose a label l_i (for any $i \in I$), and send it to $\mathbf{p}$ together with a payload of type U_i; then, the channel must be used as S_i. The labels of branch/select types are all distinct and their order is irrelevant. We omit $\&/\oplus$ when I is a singleton: $\mathbf{p}!l_1(\mathtt{Int}).S_1$ stands for $\mathbf{p} \oplus_{i\in 1} !l_i(\mathtt{Int}).S_i$. The **recursive type** $\mu \mathtt{t}.S$ and **type variable** $\mathtt{t}$ model guarded recursion. $\mathtt{end}$ is the type of a **terminated channel**. **Base types** $B, B', \ldots$ can be types like $\mathtt{Bool}$, $\mathtt{Int}$, etc **Payload types** $U, U_i, \ldots$ are either base types, or *closed* session types.

The relationship between global types and session types is formalised by the notion of projection.

Definition 3.3.1. The projection of G onto a role $\mathbf{q}$, written $G \upharpoonright \mathbf{q}$, is:

$$\mathtt{end} \upharpoonright \mathbf{q} \triangleq \mathtt{end} \qquad \mathtt{t} \upharpoonright \mathbf{q} \triangleq \mathtt{t} \qquad (\mu \mathtt{t}.G) \upharpoonright \mathbf{q} \triangleq \begin{cases} \mu \mathtt{t}.(G \upharpoonright \mathbf{q}) & \text{if } G \upharpoonright \mathbf{q} \neq \mathtt{t}' \, (\forall \mathtt{t}') \\ \mathtt{end} & \text{otherwise} \end{cases}$$

$$(\mathtt{p} \to \mathtt{p}' \colon \{l_i(U_i).G_i\}_{i\in I}) \upharpoonright \mathbf{q} \triangleq \begin{cases} \mathbf{p}' \oplus_{i\in I} !l_i(U_i).(G_i \upharpoonright \mathbf{q}) & \text{if } \mathbf{q} = \mathbf{p}, \\ \mathbf{p} \,\&_{i\in I}\, ?l_i(U_i).(G_i \upharpoonright \mathbf{q}) & \text{if } \mathbf{q} = \mathbf{p}', \\ \bigsqcap_{i\in I}(G_i \upharpoonright \mathbf{q}) & \text{if } \mathbf{p} \neq \mathbf{q} \neq \mathbf{p}' \end{cases}$$

Where the *merge operator for session types*, $\bigsqcap$, is defined by:

$$\mathtt{end} \sqcap \mathtt{end} \triangleq \mathtt{end} \qquad \mathtt{t} \sqcap \mathtt{t} \triangleq \mathtt{t} \qquad \mu t.S \sqcap \mu t.S' \triangleq \mu t.(S \sqcap S')$$

$$\mathbf{p} \,\&_{i\in I}\, ?l_i(U_i).S_i \sqcap \mathbf{p} \,\&_{j\in J}\, ?l_j(U_j).S'_j \triangleq \mathbf{p} \,\&_{k\in I\cap J}\, ?l_k(U_k).(S_k \sqcap S'_k) \,\&\, \mathbf{p} \,\&_{j\in J\setminus I}\, ?l_j(U_j).S_i \,\&\, \mathbf{p} \,\&_{j\in J\setminus I}\, ?l_j(U_j).S'_j$$

$$\mathbf{p} \oplus_{i\in I} !l_i(U_i).S_i \sqcap \mathbf{p} \oplus_{i\in I} !l_i(U_i).S_i \triangleq \mathbf{p} \oplus_{i\in I} !l_i(U_i).S_i$$

Projecting $\mathtt{end}$ or a type variable $\mathtt{t}$ onto any role does not change it. Projecting a recursive type $\mu \mathtt{t}.G$ onto $\mathbf{q}$ means projecting G onto $\mathbf{q}$. However, if G does not involve $\mathbf{q}$ then $G \upharpoonright \mathbf{q}$ is a type variable, $\mathtt{t}'$, and it must be replaced by $\mathtt{end}$ to avoid introducing an unguarded recursive type. Projecting an interaction between $\mathbf{p}$ and $\mathbf{p}'$ onto either $\mathbf{p}$ or $\mathbf{p}'$ produces a select or a branch. Projecting onto a different role $\mathbf{q}$ ignores the interaction and combines the projections of the continuations using the merge operator.

The merge operator, $\sqcap$, introduced in [46, 144], allows more global types to have defined projections, which in turn allows more processes to be typed. Different external choices from the same role $\mathbf{p}$ are integrated by merging the continuation types following a common message label, and including the branches with different labels. Merging for internal choices is undefined unless the interactions are identical. This excludes meaningless types that result when a sender $\mathbf{p}$ is unaware of which branch has been chosen by other roles in a previous interaction.

Definition 3.3.2. For a session type S, $\mathtt{roles}(S)$ denotes the set of roles occurring in S. We write $\mathbf{p} \in S$ for $\mathbf{p} \in \mathtt{roles}(S)$, and $\mathbf{p} \in S \setminus \mathbf{q}$ for $\mathbf{p} \in \mathtt{roles}(S) \setminus \{\mathbf{q}\}$.

Partial session types Figure 3.7 have the same cases as local types, without role annotations. Partial types have a notion of *duality* which exchanges branch and select but preserves payload types.

Definition 3.3.3. $\overline{H}$ is the *dual of* H, defined by:

$$\overline{\oplus_{i\in I}!l_i(U_i).H_i} \triangleq \&_{i\in I}\,?l_i(U_i).\overline{H_i} \qquad \overline{\&_{i\in I}?l_i(U_i).H_i} \triangleq \oplus_{i\in I}\,!l_i(U_i).\overline{H_i}$$

$$\overline{\mathtt{end}} \triangleq \mathtt{end} \qquad \overline{\mathtt{t}} \triangleq \mathtt{t} \qquad \overline{\mu t.H} \triangleq \mu t.\overline{H}$$

Similarly to the projection of global types to local types, a local type can be projected onto a role $\mathbf{q}$ to give a partial type. This yields a partial type that only describes the communications in S that involve $\mathbf{q}$. The definition follows the same principles as the previous definition (cf. Definition 3.3.1).

Definition 3.3.4. $S \upharpoonright \mathbf{q}$ is the *partial projection of* S *onto* $\mathbf{q}$:

$$\mathtt{end}\upharpoonright\mathbf{q} \triangleq \mathtt{end} \qquad \mathtt{t}\upharpoonright\mathbf{q} \triangleq \mathtt{t} \qquad (\mu\mathtt{t}.S)\upharpoonright\mathbf{q} \triangleq \begin{cases} \mu\,\mathtt{t}.(S\upharpoonright\mathbf{q}) & \text{if } S\upharpoonright\mathbf{q} \neq \mathtt{t}'\,(\forall \mathtt{t}') \\ \mathtt{end} & \text{otherwise} \end{cases}$$

$$(\mathbf{p}\,\oplus_{i\in I}!l_i(U_i).S_i)\upharpoonright\mathbf{q} \triangleq \begin{cases} \oplus_{i\in I}\,!l_i(U_i).(S_i\upharpoonright\mathbf{q}) & \text{if } \mathbf{q}=\mathbf{p}, \\ \bigsqcap_{i\in I}(S_i\upharpoonright\mathbf{q}) & \text{if } \mathbf{p}\neq\mathbf{q} \end{cases}$$

$$(\mathbf{p}\,\&_{i\in I}?l_i(U_i).S_i)\upharpoonright\mathbf{q} \triangleq \begin{cases} \&_{i\in I}\,?l_i(U_i).S_i\upharpoonright\mathbf{q} & \text{if } \mathbf{q}=\mathbf{p}, \\ \bigsqcap_{i\in I}(S_i\upharpoonright\mathbf{q}) & \text{if } \mathbf{p}\neq\mathbf{q} \end{cases}$$

Where the *merge operator for partial session types*, $\bigsqcap$, is defined by:

$$\mathtt{end}\sqcap\mathtt{end} \triangleq \mathtt{end} \qquad \mathtt{t}\sqcap\mathtt{t} \triangleq \mathtt{t} \qquad \mu t.H \sqcap \mu t.H' \triangleq \mu t.(H\sqcap H')$$

$$\&_{i\in I}\,?l_i(U_i).H_i \sqcap \&_{i\in I}\,?l_i(U_i).H'_i \triangleq \&_{i\in I}\,?l_i(U_i).(H_i\sqcap H'_i)$$

$$\oplus_{i\in I}\,!l_i(U_i).H_i \sqcap \oplus_{j\in J}\,!l_j(U_j).H'_j \triangleq \left(\oplus_{k\in I\cap J}\,!l_k(U_k).(H_k\sqcap H'_k)\right) \oplus \left(\oplus_{i\in I\setminus J}\,!l_i(U_i).H_i\right) \oplus \left(\oplus_{j\in J\setminus I}\,!l_j(U_j).H'_j\right)$$

Unlike session type merging, $\sqcap$ can combine different *internal* choices, but *not* external choices because that could violate type safety. Different internal choices can depend on the outcome of previous interactions with other roles, since the dependency can be safely approximated as an internal choice. Different external choices, however cannot capture this dependency.

Definition 3.3.5 (Subtyping)**.** *Subtyping on session types* $\leqslant$ is the largest relation such that:

(i) if $S \leqslant S'$, then $\forall \mathbf{p} \in (\mathtt{roles}(S) \cup \mathtt{roles}(S'))$ $S\upharpoonright\mathbf{p} \leqslant S'\upharpoonright\mathbf{p}$, and

(ii) is closed backwards under the coinductive rules in Figure 3.8.

Subtyping on partial session types $\leqslant$ is defined coinductively by the rules in Figure 3.9.

$$[\text{SBR}]\ \frac{\forall i \in I \qquad U_i \leqslant U_i' \qquad S_i \leqslant S_i'}{\mathbf{p}\ \&_{i\in I} ?l_i(U_i).S_i \leqslant \mathbf{p}\ \&_{i\in I\cup J}\ ?l_i(U_i').S_i'} \qquad [\text{SSEL}]\ \frac{\forall i \in I \qquad U_i' \leqslant U_i \qquad S_i \leqslant S_i'}{\mathbf{p}\ \oplus_{i\in I\cup J}\ !l_i(U_i).S_i \leqslant \mathbf{p}\ \oplus_{i\in I}\ !l_i(U_i').S_i'}$$

$$[\text{SB}]\ \frac{B \leqslant B'}{B \leqslant B'} \qquad [\text{SEND}]\ \frac{}{\mathtt{end} \leqslant \mathtt{end}} \qquad [\text{S}\mu\text{L}]\ \frac{S\{\mu \mathtt{t}.S/\mathtt{t}\} \leqslant S'}{\mu \mathtt{t}.S \leqslant S'} \qquad [\text{S}\mu\text{R}]\ \frac{S \leqslant S'\{\mu \mathtt{t}.S'/\mathtt{t}\}}{S \leqslant \mu \mathtt{t}.S'}$$

Figure 3.8: Subtyping for local session types

$$[\text{SPARBR}]\ \frac{\forall i \in I \qquad U_i \leqslant U_i' \qquad H_i \leqslant H_i'}{\&_{i\in I} ?l_i(U_i).H_i \leqslant \&_{i\in I\cup J}\ ?l_i(U_i').H_i'} \qquad [\text{SPARSEL}]\ \frac{\forall i \in I \qquad U_i' \leqslant U_i \qquad H_i \leqslant H_i'}{\oplus_{i\in I\cup J}\ !l_i(U_i).H_i \leqslant \oplus_{i\in I}\ !l_i(U_i').H_i'}$$

$$[\text{SPAREND}]\ \frac{}{\mathtt{end} \leqslant \mathtt{end}} \qquad [\text{SPAR}\mu\text{L}]\ \frac{H\{\mu \mathtt{t}.H/\mathtt{t}\} \leqslant H'}{\mu \mathtt{t}.H \leqslant H'} \qquad [\text{SPAR}\mu\text{R}]\ \frac{H \leqslant H'\{\mu \mathtt{t}.H'/\mathtt{t}\}}{H \leqslant \mu \mathtt{t}.H'}$$

Figure 3.9: Subtyping for partial session types

The first clause of the definition *(i)* links local and partial subtyping, and ensures that if two types are related, then their partial projections exist. This clause is used later in defining consistency in Definition 3.4.2. In the second clause *(ii)* rules [SBR], [SSEL] define subtyping on branching and selection types respectively. [SBR] is covariant (preserves the ordering of types) both in the continuation types: $S_i \leqslant S_i'$ and in the number of branches offered $U_i \leqslant U_i'$. [SSEL] is covariant in the continuation types: $S_i \leqslant S_i'$ and contravariant (reverses the ordering of types) in the number of choices: $U_i' \leqslant U_i$.

[SB] relates base types, if they are related by $\leqslant$. [SEND] relates terminated channel types. [SμL] and [SμR] are standard under coinduction [116, § 21], relating types up-to their unfolding.

3.4 Typing

Definition 3.4.1 (Typing Contexts)**.** Δ denotes a partial mapping from process variables to n-tuples of types, and Γ denotes a partial mapping from channels to types, both defined in Figure 3.7.

The composition Γ_1, Γ_2 is defined if and only if $\mathrm{dom}(\Gamma_1) \cap \mathrm{dom}(\Gamma_2) = \varnothing$.

We write $s \notin \Gamma$ if and only if $\forall \mathbf{p} : \mathtt{s}[\mathbf{p}] \notin \mathrm{dom}(\Gamma)$, when session s does not occur in Γ.

We write $\mathrm{dom}(\Gamma) = s$ if and only if $\forall \mathtt{c} \in \mathrm{dom}(\Gamma)$ there is $\mathbf{p}$ such that $\mathtt{c} = \mathtt{s}[\mathbf{p}]$, when Γ only contains session s.

We write $\Gamma \leqslant \Gamma'$ if and only if $\mathrm{dom}(\Gamma) = \mathrm{dom}(\Gamma')$ and $\forall \mathtt{c} \in \mathrm{dom}(\Gamma) : \Gamma(\mathtt{c}) \leqslant \Gamma'(\mathtt{c})$.

Similar to binary session types, we say that Γ *is unrestricted*, $\mathtt{un}(\Gamma)$, if and only if for all $\mathtt{c} \in \mathrm{dom}(\Gamma)$, $\Gamma(\mathtt{c})$ is either a base type or $\mathtt{end}$.

The *typing contexts composition* $\circ$ is the commutative operator with $\varnothing$ as neutral element:

$$[\text{TVAR}]\ \frac{\text{un}(\Gamma)}{\Gamma, \mathtt{c} : S \vdash \mathtt{c} : S} \qquad [\text{TVAL}]\ \frac{\text{un}(\Gamma) \qquad v \in B}{\Gamma \vdash v : B} \qquad [\text{TINACT}]\ \frac{\text{un}(\Gamma)}{\Delta; \Gamma \vdash \mathbf{0}}$$

$$[\text{TPAR}]\ \frac{\Delta; \Gamma_1 \vdash P \quad \Delta; \Gamma_2 \vdash Q}{\Delta; \Gamma_1 \circ \Gamma_2 \vdash P \mid Q} \qquad [\text{TSUB}]\ \frac{\Delta; \Gamma, \mathtt{c} : U \vdash P \qquad U' \leqslant U}{\Delta; \Gamma, \mathtt{c} : U' \vdash P}$$

$$[\text{TRES}]\ \frac{\Delta; \Gamma, \Gamma' \vdash P \qquad \Gamma' = \{\mathtt{s}[\mathbf{p}] : S_{\mathbf{p}}\}_{\mathbf{p} \in I} \text{ complete}}{\Delta; \Gamma \vdash (\nu\, s : \Gamma') P}$$

$$[\text{TSEL}]\ \frac{\Gamma_1 \vdash v : U_j \qquad \Delta; \Gamma_2, \mathtt{c} : S_j \vdash P \qquad j \in I}{\Delta; \Gamma_1 \circ \Gamma_2, \mathtt{c} : \mathbf{p} \oplus_{i \in I} !l_i(U_i).S_i \vdash \mathtt{c}[\mathbf{p}] \oplus \langle l_j(v) \rangle.P}$$

$$[\text{TBR}]\ \frac{\Delta; \Gamma, x_i : U_i, \mathtt{c} : S_i \vdash P_i \qquad \forall i \in I}{\Delta; \Gamma, \mathtt{c} : \mathbf{p}\ \&_{i \in I} ?l_i(U_i).S_i \vdash \mathtt{c}[\mathbf{p}] \&_{i \in I} \{l_i(x_i).P_i\}}$$

$$[\text{TDEF}]\ \frac{\Delta, X : \tilde{U}; \tilde{x} : \tilde{U} \vdash P \qquad \Delta, X : \tilde{U}; \Gamma \vdash Q}{\Delta; \Gamma \vdash \mathbf{def}\ X\langle \tilde{x} : \tilde{U} \rangle = P\ \mathbf{in}\ Q}$$

$$[\text{TCALL}]\ \frac{\forall i \in \{1..n\} \qquad \Gamma \vdash v_i : U_i \qquad \text{un}(\Gamma)}{\Delta, X : U_1, \ldots, U_n; \Gamma \vdash X\langle v_1, \ldots, v_n \rangle}$$

Figure 3.10: Typing rules for the π-calculus with multiparty sessions

$$\Gamma_1, \mathtt{c} : U \circ \Gamma_2, \mathtt{c}_i : U_i \triangleq (\Gamma_1 \circ \Gamma_2), \mathtt{c} : U, \mathtt{c}_i : U_i \quad (\text{if } \text{dom}(\Gamma_2) \not\ni \mathtt{c} \neq \mathtt{c}_i \notin \text{dom}(\Gamma_1))$$
$$\Gamma_1, x : B \circ \Gamma_2, x : B \triangleq (\Gamma_1 \circ \Gamma_2), x : B$$

Definition 3.4.2 (Completeness and consistency)**.**
Γ is *complete* if and only if for all $\mathtt{s}[\mathbf{p}] : S_{\mathbf{p}} \in \Gamma$, $\mathbf{q} \in S_{\mathbf{p}}$ implies $\mathtt{s}[\mathbf{q}] : S_{\mathbf{q}} \in \text{dom}(\Gamma)$.
Γ is *consistent* if and only if for all $\mathtt{s}[\mathbf{p}] : S_{\mathbf{p}}, \mathtt{s}[\mathbf{q}] : S_{\mathbf{q}} \in \Gamma$, with $\mathbf{p} \neq \mathbf{q}$ we have $\overline{S_{\mathbf{p}} \upharpoonright \mathbf{q}} \leqslant S_{\mathbf{q}} \upharpoonright \mathbf{p}$.

Completeness means that if a channel is in Γ, then Γ also contains the other endpoints of the channel. In this case, there is a self-contained collection of channels that can communicate. **Consistency** means that the opposite endpoints of every channel have dual partial types. Consistency of the session typing context Γ is necessary to prove subject reduction. An important note is that the consistency definition Definition 3.4.2, introduced by Scalas *et al.*in [122], uses subtyping $\leqslant$ rather than syntactic type equality $=$ to relate dual partial projections, and thus allows subject reduction to hold. It is a weaker definition than the one usually found in the literature, and fixes a long-standing mistake in subject reduction proofs appearing in [45, 46, 145].

Definition 3.4.3 (Typing judgements)**.** Typing judgements are inductively defined by the rules in Figure 3.10. Typing judgements have the form: $\Delta; \Gamma \vdash P$ with Γ consistent, and $\forall \mathtt{c} : S \in \Gamma, S \upharpoonright \mathbf{p}$ is defined $\forall \mathbf{p} \in S$. Δ is omitted when empty.

[TVAR] says that a channel has the type assumed in the session typing context. [TVAL] relates base values to their type. [TINACT] states that the terminated process is well typed in any unrestricted typing context, that is when all channels in Γ have $\mathtt{end}$ type. [TPAR] states that the parallel composition of P and Q is well typed under the composition of the corresponding typing contexts. [TSUB] is a standard subsumption rule using $\leqslant$ (Definition 3.3.5). [TRES] requires the restricted environment Γ' to be *complete* (Definition 3.4.2) and the open process P to be well typed under Γ, Γ'. [TSEL] (resp. [TBR]) states that the selection (resp. branching) on channel $\mathtt{c}[\mathbf{p}]$ is well typed if the channel has compatible select(resp. branching) type and the continuations $P_i, \forall i \in I$ are well-typed with the continuation session types. By [TDEF] $\mathbf{def}\, X\langle \tilde{x} : \tilde{U} \rangle = P \,\mathbf{in}\, Q$ is well-typed if P uses the arguments $\tilde{x}$ according to $\tilde{U}$, and if $X : \tilde{U}$ when typing both P and Q. By [TCALL] $X\langle v_1, \ldots, v_n \rangle$ is well-typed if the types of $\tilde{v}$ match those of the parameters of X, and if Γ is unrestricted. This last assumption ensures that channels requiring more inputs/outputs cannot be forgotten, thus preserving linearity. Together, rules [TDEF] and [TCALL] allow us to model recursive processes.

3.5 Main Results

In this section we present the main properties satisfied by the session type system presented. The following lemmas and theorems are proven in [122].

Definition 3.5.1 (Typing context reduction)**.** The *reduction* $\Gamma \to \Gamma'$ is:

$$\mathtt{s}[\mathbf{p}] : S_{\mathbf{p}}, \mathtt{s}[\mathbf{q}] : S_{\mathbf{q}} \;\to\; \mathtt{s}[\mathbf{p}] : S_k, \mathtt{s}[\mathbf{q}] : S'_k \quad \text{if} \begin{cases} S_{\mathbf{p}} = \mathbf{q} \oplus_{i \in I} !l_i(U_i).S_i & k \in I \\ \mathtt{unf}(S_{\mathbf{q}}) = \mathbf{p} \,\&_{i \in I \cup J}\, ?l_i(U'_i).S'_i & U_k \leqslant U'_k \end{cases}$$

$$\Gamma, \mathtt{c} : U \;\to\; \Gamma', \mathtt{c} : U' \qquad \text{if } \Gamma \to \Gamma' \text{ and } U \leqslant U'$$

Definition 3.5.1 accommodates subtyping (hence, uses $\leqslant$) and iso-recursive type equality, hence, unfolds types explicitly.

Theorem 3.5.1 (Subject reduction)**.** *If* $\Delta; \Gamma \vdash P$ *and* $P \longrightarrow P'$*, then* $\exists \Gamma'$*:* $\Gamma \longrightarrow^* \Gamma'$ *and* $\Delta; \Gamma' \vdash P'$*.*

Theorem 3.5.2 (Deadlock freedom)**.** *Let* $\varnothing \cdot \varnothing \vdash P$*, where* $P \equiv (\nu\, s : G)\big|_{i \in I} P_i$ *and each* P_i only interacts on $\mathtt{s}[\mathbf{p_i}]$*. Then,* P *is* deadlock-free*: i.e.,* $P \longrightarrow^* P' \not\to$ *implies* $P' \equiv \mathbf{0}$*.*

3.6 Implementations

At the heart of many implementations into mainstream programming languages lies Scribble [69, 146], a protocol description language, based on the theory of multiparty session types. A protocol in Scribble specifies the interactions between each participant in the system in a top-down manner. The Scribble tool first verifies that this global protocol is well-formed and thus describes a safe protocol; then projects the global protocol into local protocols for each role. Originally verification was achieved using fairly conservative syntactic checks, however recent work [78] takes a more semantic approach through the use of 1-bounded model checking, thus being less restrictive with the interactions that can be represented.

Once the global protocol has been found to be well-formed, it can be projected to a set of local protocols. A local Scribble protocol is generated for each role declared in the definition of the global protocol. Local protocols correspond to local session types, they describe interactions from the viewpoint of a single entity. They can be used directly by a type checker, or via runtime verification techniques to verify that an endpoint code implementation complies to the interactions prescribed by the specification.

Static Checking. One of the earliest implementations of statically-checked multiparty session types is Multiparty Session C [103], which implements multiparty session types in C via a lightweight runtime system and a compiler plugin. Multiparty Session C concentrates on bringing the benefits of multiparty session types to the domain of high-performance computing. Later work by Ng *et al.* [102] uses Scribble to generate MPI backbone code, reducing the amount of boilerplate a developer of HPC applications must write, and guaranteeing deadlock-freedom.

Hu and Yoshida [77] describe an approach called endpoint API generation, where local types guide the generation of state channel objects for each role. State channels guide a developer in following the protocol through the use of an object-oriented call-chaining API. Linearity is enforced dynamically through a simple run-time check, ensuring that each state channel object is used only once.

Typestate [127] is a related concept to session types, as shown in work by Gay *et al.* [64]. This relation is exploited in the work of Kouzapas *et al.* [85, 86] to define a typestate system for Java based on multiparty session types. The work describes the design and implementation of two tools, Mungo and StMungo. Mungo extends Java with typestate definitions, by allowing classes to be associated with a definition of the permitted sequences of method calls. The second, StMungo ("Scribble-to-Mungo") translates from Scribble local protocols into typestate specifications and Java program skeletons, which can be further implemented, and checked by Mungo to ensure that the implementation still follows the protocol.

Type providers [114] allow statically-typed access to unstructured and untyped external data sources such as CSV roles and SQL schemas via compile-time metaprogramming. Neykova *et al.* [98] leverage the work on endpoint API generation to define a session type provider, extending

type providers to the domain of communication-centric software and introducing interaction refinements: predicates on message payloads which are enforced by use of an SMT solver. Scalas *et al.* [122] extend the continuation-passing translation from binary session types into the linear π-calculus introduced by Kobayashi [84] and later extended by Dardha *et al.* [40] to the multiparty setting. Their approach lends itself to an implementation of multiparty session types in Scala following previous work on lchannels [124], in particular being the first work to support distributed delegation in the multiparty setting.

Runtime Monitoring. An alternative approach to checking conformance to protocols statically is to verify conformance at *run-time*. Deniélou and Yoshida [45] describe deep connections between multiparty session types and communicating finite-state automata [18], identifying a class of communicating finite-state automata called multiparty session automata, which enjoy safety properties such as deadlock-freedom.

Multiparty session automata can be used as monitors to dynamically enforce compliance with a session at run-time. Chen *et al.* [29] and Bocchi *et al.* [15] describe the theory of run-time monitoring of communication against session types. The formalism consists of an unmonitored semantics; a labelled transition system semantics of monitors; and a monitored semantics where actions are predicated on labels emitted by monitor reduction. The key results are of safety and transparency: safety means that the processes behave in accordance with the global specification, and transparency means that a monitored network behaves exactly the same as an equivalent unmonitored network conforming to the specification. SPY [101] is the first implementation of multiparty session types in a dynamically-checked programming language, implementing a Python API for session programming where communication safety is guaranteed through runtime monitors generated from Scribble specifications. Demangeon *et al.* [76] extend this work with an interruptible construct, allowing blocks to be interrupted by incoming messages.

Neykova and Yoshida [99] are the first to integrate multiparty session types and the actor model via dynamic monitoring. In the conceptual framework proposed by the authors, each actor is an entity which may take part in multiple sessions, and where a message received in one session may trigger a message to be sent in another session. The framework is implemented in Python, and communication between actors is mediated via monitors derived from Scribble specifications. Subsequent work [55] implements an extended version of Neykova and Yoshida's conceptual framework in Erlang, motivating the use of subsessions [44] to allow parts of a protocol to be repeated with new participants. Neykova and Yoshida [100] investigate failure recovery strategies in Erlang, using information gained from protocols to compute and revert to safe states when a failure occurs.

Part II

Resource Sharing via Capability-Based Multiparty Session Types

It is standard for multiparty session type systems to use access control based on *linear* or *affine* types. While useful in offering strong guarantees of communication safety and session fidelity, linearity and affinity run into the well-known problem of inflexible programming, excluding scenarios that make use of shared channels or need to store channels in shared data structures.

In this part, we present *capability-based resource sharing* for multiparty session types. In this setting, channels are split into two entities, the channel itself and the capability of using it. This gives rise to a more flexible session type system, which allows channel references to be shared and stored in persistent data structures. We prove that the resulting language satisfies type safety, we illustrate our type system through a producer-consumer case study, and conclude by discussing related work and future directions.

Chapter 4

Resource Sharing via Capability-Based Multiparty Session Types

4.1 Introduction

Session type systems must control *aliasing* when it comes to the endpoints of communication channels, to avoid race conditions. Aliasing happens when multiple channel references point to the same communication channel. This can cause issues because the type of the channel determines the type of messages that can be sent or received, and if two or more channels have different types, it can lead to errors or unexpected behaviour. For examples, if two processes P and Q both think they are running the client side of a protocol with the same server S, then a message sent by P advances the session state without Q's knowledge, which interferes with Q's attempt to run the protocol.

In order to guarantee unique ownership of channel endpoints and eliminate aliasing, most session type systems use strict *linear typing*. For more flexibility, some others use *affine typing*, which allows channels to be discarded, but they still forbid aliasing. It is possible to allow a session-typed channel to become shareable in the special case in which the session type reaches a point which is essentially stateless. However, in such systems, channels are *linearly typed* for the most interesting parts of their lifetimes—we discuss more about these possibilities in **??**.

To give a more flexible approach to resource sharing and access control, we propose a system of multiparty session types that includes techniques from the Capability Calculus [37], and from Walker *et al.*'s work on alias types [140]. The key idea is to split a communication channel into two entities: (1) the channel itself, and (2) its usage *capability*. Both entities are first-class and can be referred to separately. Channels can now be shared, or stored in shared data structures, and aliasing is allowed. However, in order to guarantee communication safety and session fidelity, i.e., type safety, capabilities are used linearly so that only one alias can be used at one time.

This approach has several benefits, and improves on the state of the art:

i) it is now possible for a system to have a communication structure defined by shared

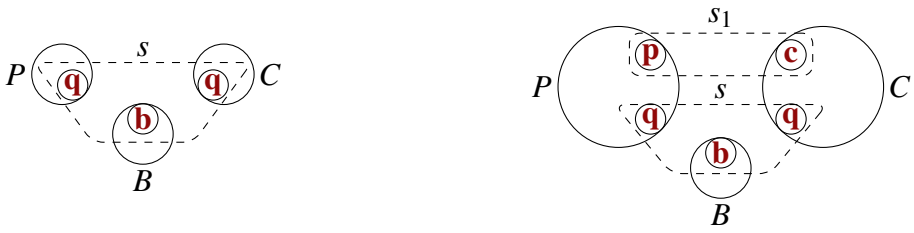

Figure 4.1: Producer-consumer system: Producer—P and Consumer—C sharing access to Buffer—B by implementing the same role $\mathbf{q}$.

(a) P and C communicate with B in session s; (b) P and C exchange the capability to use channel $\mathtt{s}[\mathbf{q}]$ in session s_1 and then use it to communicate with B in session s

channels, with the capabilities being transferred from process to process as required;

ii) a capability can be implemented as a simple token, whereas delegation of channels requires a relatively complex implementation, thus making linearity of capabilities more lightweight than linearity of channels.

Example 1 (Producer-Consumer)**.** *Producer* P *and consumer* C *communicate via buffer* B *in session* s*, given in fig. 4.1 (a).* P *and* C *implement* role $\mathbf{q}$*, and* B *implements role* $\mathbf{b}$. Shared access *to buffer* B *is captured by the fact that both* P *and* C *implement the same role* $\mathbf{q}$ *and use the same channel* $\mathtt{s}[\mathbf{q}]$ *to communicate with* B.

Following multiparty session type theory, we start by defining a global type*, describing communications among* all *participants:*

$$G_0 = \mathbf{q}\to\mathbf{b}\colon \mathsf{add}(\mathtt{Int}).\ \mathbf{q}\to\mathbf{b}\colon \mathsf{req}().\ \mathbf{b}\to\mathbf{q}\colon \mathsf{snd}(\mathtt{Int}).\ G_0$$

In G_0*, protocol proceeds as follows:* P *(playing* $\mathbf{q}$*) sends an* add *message to* B *(playing* $\mathbf{b}$*), to add data. In sequence,* C *(playing* $\mathbf{q}$*) sends a* req *message to* B*, asking for data.* B *replies with a* snd *message, sending data to* C *(playing* $\mathbf{q}$*), and the protocol repeats as* G_0. Projecting *the global protocol to each role gives us a local session type. In particular, for* B*, implementing role* $\mathbf{b}$*, we obtain:*

$$S_{\mathbf{b}} = \mathbf{q}?\mathsf{add}(\mathtt{Int}).\ \mathbf{q}?\mathsf{req}().\ \mathbf{q}!\mathsf{snd}(\mathtt{Int}).\ S_{\mathbf{b}}$$

where the $\mathbf{q}$ *annotations show the other role participating in each interaction. For the shared access by* P *and* C *(role* $\mathbf{q}$*), we obtain:*

$$S_{\mathbf{q}} = \mathbf{b}!\mathsf{add}(\mathtt{Int}).\ S'_{\mathbf{q}} \qquad S'_{\mathbf{q}} = \mathbf{b}!\mathsf{req}().\ \mathbf{b}?\mathsf{snd}(\mathtt{Int}).\ S_{\mathbf{q}}$$

Finally, the definitions of processes are as follows—we will detail the syntax in Section 4.2.

$$\begin{aligned}
\mathsf{P}\langle v\rangle &= \mathtt{s}[\mathbf{q}][\mathbf{b}]\oplus\langle\mathsf{add}(v)\rangle.\mathsf{P}\langle v+1\rangle\\
\mathsf{C}\langle\rangle &= \mathtt{s}[\mathbf{q}][\mathbf{b}]\oplus\langle\mathsf{req}()\rangle.\mathtt{s}[\mathbf{b}][\mathbf{q}]\&\{\mathsf{snd}(\mathsf{i})\}.\mathsf{C}\langle\rangle\\
\mathsf{B}\langle\rangle &= \mathtt{s}[\mathbf{q}][\mathbf{b}]\&\{\mathsf{add}(x)\}.\mathtt{s}[\mathbf{q}][\mathbf{b}]\&\{\mathsf{req}()\}.\mathtt{s}[\mathbf{b}][\mathbf{q}]\oplus\langle\mathsf{snd}(x)\rangle.\mathsf{B}\langle\rangle
\end{aligned}$$

The system of processes above is not typable using standard multiparty session type systems because role $\mathbf{q}$ *is shared by* P *and* C*, thus violating linearity of channel* $\mathtt{s}[\mathbf{q}]$*. To solve this issue while allowing sharing and aliasing, instead of associating a channel* c *with a session type* S*, we separately associate* c *with a* tracked *type* $\mathtt{tr}(\rho)$*, and* S *with* capability ρ*,* $\{\rho \mapsto S\}$*. The capability can be passed between* P *and* C *as they take turns in using the channel by sending a message* $\mathsf{turn}(\mathtt{tr}(\rho_{\mathbf{q}}))$ *containing the capability for using the channel, as illustrated in Figure 4.1 (b). As a first attempt, we now define the following global type, getting us closer to our framework.*

$$\begin{aligned} G_1 \;=\; & \mathbf{q}\to\mathbf{b}\colon \mathsf{add}(\mathtt{Int}).\ \mathbf{p}\to\mathbf{c}\colon \mathsf{turn}(\mathtt{tr}(\rho_{\mathbf{q}})).\ \mathbf{q}\to\mathbf{b}\colon \mathsf{req}().\\ & \mathbf{b}\to\mathbf{q}\colon \mathsf{snd}(\mathtt{Int}).\ \mathbf{c}\to\mathbf{p}\colon \mathsf{turn}(\mathtt{tr}(\rho_{\mathbf{q}})).G_1 \end{aligned}$$

However, a type such as $\mathtt{tr}(\rho_{\mathbf{q}})$ *is usually too specific because it refers to the capability of a particular channel. It is preferable to be able to give definitions that abstract away from specific channels. We therefore introduce* existential types*, in the style of [140], which package a channel with its capability, in the form* $\exists[\rho|\{\rho \mapsto S\}].\mathtt{tr}(\rho)$*.*

With the existential types in place, we can define our global type G *in the following way. It now includes an extra initial message from* P *to* C *containing the channel used with the buffer. The session types* $S_{\mathbf{q}}$ *and* $S'_{\mathbf{q}}$ *are the same as before. We show the interaction in Figure 4.2.*

$$\begin{aligned} G = \; & \mathbf{p}\to\mathbf{c}\colon \mathsf{buff}(\exists[\rho_{\mathbf{q}}|\{\rho_{\mathbf{q}} \mapsto S'_{\mathbf{q}}\}].\mathtt{tr}(\rho_{\mathbf{q}})).\mu\mathtt{t}.\mathbf{q}\to\mathbf{b}\colon \mathsf{add}(\mathtt{Int}).\mathbf{p}\to\mathbf{c}\colon \mathsf{turn}(\{\rho_{\mathbf{q}} \mapsto S'_{\mathbf{q}}\}).\\ & \mathbf{q}\to\mathbf{b}\colon \mathsf{req}().\mathbf{b}\to\mathbf{q}\colon \mathsf{snd}(\mathtt{Int}).\mathbf{c}\to\mathbf{p}\colon \mathsf{turn}(\{\rho_{\mathbf{q}} \mapsto S_{\mathbf{q}}\}).\mathtt{t} \end{aligned}$$

In Chapter 5 we complete this example by showing the projections to a local type for each role, and the definitions of processes that implement each role. □

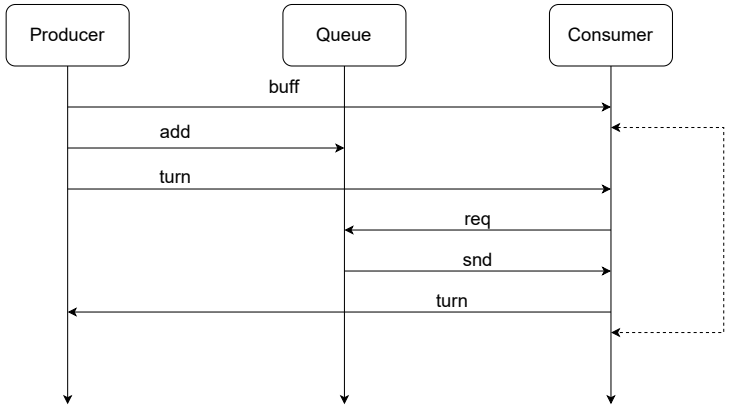

Figure 4.2: Producer-consumer Protocol

4.2 Syntax and Semantics

A message in session $\mathtt{s}$ from role $\mathbf{p}$ to role $\mathbf{q}$ has the prefix $\mathtt{s}[\mathbf{p}][\mathbf{q}]$, where $\mathtt{s}[\mathbf{p}]$ is represented by $\mathtt{c}$ in the grammar. The select and branch operations come in two forms. The first form is standard,

P	$::=$	$\mathbf{0}$	inaction
	$\mid$	$P \mid Q$	parallel composition
	$\mid$	$(\nu\, s)P$	restriction
	$\mid$	$\mathsf{c}[\mathbf{p}] \oplus \langle l(v) \rangle . P$	select
	$\mid$	$\mathsf{c}[\mathbf{p}] \& _{i \in I} \{ l_i(x_i) . P_i \}$	branch
	$\mid$	$\mathsf{c}[\mathbf{p}] \oplus \langle l(\mathsf{pack}(\rho, \mathsf{s}[\mathbf{q}])) \rangle . P$	select pack
	$\mid$	$\mathsf{c}[\mathbf{p}] \& _{i \in I} \{ l_i(\mathsf{pack}(\rho_i, \mathsf{s}_\mathsf{i}[\mathbf{q}])) . P_i \}$	branch pack
	$\mid$	$\mathbf{def}\ D\ \mathbf{in}\ P$	recursion
	$\mid$	$X \langle \tilde{x} \rangle$	process call
D	$::=$	$X \langle \tilde{x} \rangle = P$	process declaration
c	$::=$	x	variable
	$\mid$	$\mathsf{s}[\mathbf{p}]$	channel with role p
v	$::=$	c	channel
	$\mid$	ρ	capability
	$\mid$	$\mathsf{true} \mid \mathsf{false} \mid 0 \mid 1 \mid \ldots$	base value

Figure 4.3: Multiparty session π-calculus with capabilities

and the second form handles *packages*, which are the novel feature of our type system. A package consists of a capability ρ and a channel of type $\mathtt{tr}(\rho)$. We will see in Section 4.3 in the typing rules, the capability is existentially quantified. This enables a channel to be delegated, with the information that it is linked to *some* capability, which will be transmitted in a second message.

We define a **reduction-based operational semantics** by the rules in Figure 4.4. Rule [RCom] is a standard communication between roles $\mathbf{p}$ and $\mathbf{q}$. Rule [RComP] is communication of an existential package. Rule [RCall] defines a standard approach to handling process definitions. The rest are standard contextual rules.

4.3 Types and Subtypes

Global types are given in Figure 4.5. To the global types introduced in Section 3.3 we add a new interaction type, *pack interaction*, $\mathtt{p} \to \mathtt{q}$: $\{ l_i(\exists [\rho_i | \{\rho_i \mapsto U_i\}].tr(\rho_i)).G_i \}_{i \in I}$. Pack interaction states that role $\mathbf{p}$ sends to role $\mathbf{q}$ one of the (pairwise distinct) labels l_i for $i \in I$, together with a package payload $\exists [\rho_i | \{\rho_i \mapsto U_i\}].tr(\rho_i)$ representing a channel with its capability. If the chosen label is l_j, then the interaction proceeds as G_j, with the capability ρ_j being bound in G_j.

Local (session) types are given in Figure 4.5. To the types introduced in Section 3.3 we add two new types, *pack select* and *pack branch*. They act in a similar manner to the classic select and branch types, but bind the capability ρ_i for the continuation type S_i. Pack branch $\mathbf{p}\ \&_{i \in I}\ ?l_i(\exists [\rho_i | \{\rho_i \mapsto U_i\}].\mathtt{tr}(\rho_i)).S_i$ describes a channel that can receive a label l_i from role $\mathbf{p}$ (for

$$[\text{RCom}]\ \frac{j \in I \text{ and } fv(v) = \varnothing}{\mathtt{s}[\mathbf{p}][\mathbf{q}]\&_{i\in I}\{l_i(x_i).P_i\} \mid \mathtt{s}[\mathbf{q}][\mathbf{p}]\oplus\langle l_j(v)\rangle.P \longrightarrow P_j\{v/x_j\} \mid P}$$

$$[\text{RComP}]\ \frac{j \in I}{\mathtt{s}[\mathbf{p}][\mathbf{q}]\&_{i\in I}\{l_i(\mathsf{pack}(\rho_i,\mathtt{v}_i)).P_i\} \mid \mathtt{s}[\mathbf{q}][\mathbf{p}]\oplus\langle l_j(\mathsf{pack}(\rho,\mathtt{v}))\rangle.P \longrightarrow P_j\{\mathtt{v}/\mathtt{v}_i\} \mid P}$$

$$[\text{RCall}]\ \frac{\widetilde{x} = x_1,\ldots,x_n \qquad \widetilde{v} = v_1,\ldots,v_n \qquad fv(\widetilde{v}) = \varnothing}{\mathbf{def}\, X\langle\widetilde{x}\rangle = P \,\mathbf{in}\, (X\langle\widetilde{x}\rangle \mid Q) \longrightarrow \mathbf{def}\, X\langle\widetilde{x}\rangle = P \,\mathbf{in}\, (P\{\widetilde{v}/\widetilde{x}\} \mid Q)}$$

$$[\text{RRes}]\ \frac{P \longrightarrow Q}{(\nu\, s)P \longrightarrow (\nu\, s)Q} \qquad [\text{RPar}]\ \frac{P \longrightarrow Q}{P \mid R \longrightarrow Q \mid R} \qquad [\text{RDef}]\ \frac{P \longrightarrow Q}{\mathbf{def}\, D \,\mathbf{in}\, P \longrightarrow \mathbf{def}\, D \,\mathbf{in}\, Q}$$

Figure 4.4: Reduction (processes)

some $i \in I$, chosen by $\mathbf{p}$), together with a *payload* of type $\exists[\rho_i|\{\rho_i \mapsto U_i\}].\mathtt{tr}(\rho_i)$, and then bind the capability ρ_i for the continuation type S_i. Pack select $\mathbf{p} \oplus_{i\in I} !l_i(\exists[\rho_i|\{\rho_i \mapsto U_i\}].\mathtt{tr}(\rho_i)).S_i$, describes a channel that can choose a label l_i (for any $i \in I$), and send it to $\mathbf{p}$ together with a payload of type $\exists[\rho_i|\{\rho_i \mapsto U_i\}].\mathtt{tr}(\rho_i)$; and then bind the capability ρ_i for the continuation type S_i.

The relationship between global types and session types is formalised by the notion of projection. We extend the definition given in Definition 3.3.1 for the pack interaction type added.

Definition 4.3.1. The projection of G onto a role $\mathbf{q}$, written $G \upharpoonright \mathbf{q}$, is:

$$\mathtt{end} \upharpoonright \mathbf{q} \triangleq \mathtt{end} \qquad \mathtt{t} \upharpoonright \mathbf{q} \triangleq \mathtt{t} \qquad (\mu\mathtt{t}.G) \upharpoonright \mathbf{q} \triangleq \begin{cases} \mu\mathtt{t}.(G \upharpoonright \mathbf{q}) & \text{if } G \upharpoonright \mathbf{q} \neq \mathtt{t}'\ (\forall \mathtt{t}') \\ \mathtt{end} & \text{otherwise} \end{cases}$$

$$(\mathtt{p}\to\mathtt{p}'\colon \{l_i(U_i).G_i\}_{i\in I}) \upharpoonright \mathbf{q} \triangleq \begin{cases} \mathbf{p}' \oplus_{i\in I} !l_i(U_i).(G_i \upharpoonright \mathbf{q}) & \text{if } \mathbf{q} = \mathbf{p}, \\ \mathbf{p} \,\&_{i\in I}\, ?l_i(U_i).(G_i \upharpoonright \mathbf{q}) & \text{if } \mathbf{q} = \mathbf{p}', \\ \bigsqcap_{i\in I}(G_i \upharpoonright \mathbf{q}) & \text{if } \mathbf{p} \neq \mathbf{q} \neq \mathbf{p}' \end{cases}$$

$$(\mathtt{p}\to\mathtt{p}'\colon \{l_i(\exists[\rho_i|\{\rho_i \mapsto U_i\}].\mathtt{tr}(\rho_i)).G_i\}_{i\in I}) \upharpoonright \mathbf{q} \triangleq \begin{cases} \mathbf{p}' \oplus_{i\in I} !l_i(\exists[\rho_i|\{\rho_i \mapsto U_i\}].\mathtt{tr}(\rho_i)).(G_i \upharpoonright \mathbf{q}) & \text{if } \mathbf{q} = \mathbf{p}, \\ \mathbf{p} \,\&_{i\in I}\, ?l_i(\exists[\rho_i|\{\rho_i \mapsto U_i\}].\mathtt{tr}(\rho_i)).(G_i \upharpoonright \mathbf{q}) & \text{if } \mathbf{q} = \mathbf{p}', \\ \bigsqcap_{i\in I}(G_i \upharpoonright \mathbf{q}) & \text{if } \mathbf{p} \neq \mathbf{q} \neq \mathbf{p}' \end{cases}$$

Where the *merge operator for session types*, $\bigsqcap$, is defined by:

Global type	$G ::= \mathtt{end}$	termination
	$\mid \mathtt{p}\to\mathtt{q}\colon \{l_i(U_i).G_i\}_{i\in I}$	interaction
	$\mid \mathtt{p}\to\mathtt{q}\colon \{l_i(\exists[\rho_i\vert\{\rho_i\mapsto U_i\}].tr(\rho_i)).G_i\}_{i\in I}$	pack interaction
	$\mid \mathtt{t}$	type variable
	$\mid \mu\mathtt{t}.G$	recursive type
Local session type	$S ::= \mathtt{end}$	terminated session
	$\mid \mathbf{p}\oplus_{i\in I}!l_i(U_i).S_i$	selection towards role p
	$\mid \mathbf{p}\,\&_{i\in I}?l_i(U_i).S_i$	branching from role p
	$\mid \mathbf{p}\oplus_{i\in I}!l_i(\exists[\rho_i\vert\{\rho_i\mapsto U_i\}].\mathtt{tr}(\rho_i)).S_i$	pack selection to role **p**
	$\mid \mathbf{p}\,\&_{i\in I}?l_i(\exists[\rho_i\vert\{\rho_i\mapsto U_i\}].\mathtt{tr}(\rho_i)).S_i$	pack branching from role **p**
	$\mid \mathtt{t}$	type variable
	$\mid \mu\mathtt{t}.S$	recursive type
Partial session type	$H ::= \mathtt{end}$	terminated session
	$\mid \oplus_{i\in I}!l_i(U_i).H_i$	selection
	$\mid \&_{i\in I}?l_i(U_i).H_i$	branching
	$\mid \oplus_{i\in I}!l_i(\exists[\rho_i\vert\{\rho_i\mapsto U_i\}].\mathtt{tr}(\rho_i)).H_i$	pack selection
	$\mid \&_{i\in I}?l_i(\exists[\rho_i\vert\{\rho_i\mapsto U_i\}].\mathtt{tr}(\rho_i)).H_i$	pack branching
	$\mid \mathtt{t}$	type variable
	$\mid \mu\,\mathtt{t}.H$	recursive type
Capabilities	$C ::= \varnothing \mid C\otimes\{\rho\mapsto S\}$	
Ground type	$B ::= \mathtt{Int} \mid \mathtt{Bool}$	
Payload type	$U ::= B$	ground type
	$\mid \mathtt{tr}(\rho)$	tracked type
	$\mid \{\rho\mapsto S\}$	capability type
	$\mid S$ closed	session type
Environment	$\Gamma ::= \varnothing \mid \Gamma, x:U \mid \Gamma, \mathtt{s}[\mathbf{p}]:\mathtt{tr}(\rho)$	
	$\Delta ::= \varnothing \mid \Delta, X:\tilde{U}$	*process names*

All branch and select types have the conditions $I \neq \emptyset$ and U_i closed.

Figure 4.5: Types, capabilities, environments

$$\mathtt{end} \sqcap \mathtt{end} \triangleq \mathtt{end} \qquad \mathtt{t} \sqcap \mathtt{t} \triangleq \mathtt{t} \qquad \mu t.S \sqcap \mu t.S' \triangleq \mu t.(S \sqcap S')$$

$$\mathbf{p} \,\&_{i\in I}\, ?l_i(U_i).S_i \sqcap \mathbf{p} \,\&_{j\in J}\, ?l_j(U_j).S'_j \triangleq \mathbf{p} \,\&_{k\in I\cap J}\, ?l_k(U_k).(S_k \sqcap S'_k) \,\&\, \mathbf{p} \,\&_{j\in I\setminus J}\, ?l_i(U_i).S_i \,\&\, \mathbf{p} \,\&_{j\in J\setminus I}\, ?l_j(U_j).S'_j$$

$$\mathbf{p} \oplus_{i\in I} !l_i(U_i).S_i \sqcap \mathbf{p} \oplus_{i\in I} !l_i(U_i).S_i \triangleq \mathbf{p} \oplus_{i\in I} !l_i(U_i).S_i$$

$$\mathbf{p} \,\&_{i\in I}\, ?l_i(\exists[\rho_i|\{\rho_i\mapsto U_i\}].\mathtt{tr}(\rho_i)).S_i \sqcap \mathbf{p} \,\&_{j\in J}\, ?l_j(\exists[\rho_j|\{\rho_j\mapsto U_j\}].\mathtt{tr}(\rho_j)).S'_j \triangleq \mathbf{p} \,\&_{k\in I\cap J}\, ?l_k(\exists[\rho_k|\{\rho_k\mapsto U_k\}].\mathtt{tr}(\rho_k)).(S_k \sqcap S'_k) \,\&\, \mathbf{p} \,\&_{j\in I\setminus J}\, ?l_i(\exists[\rho_i|\{\rho_i\mapsto U_i\}].\mathtt{tr}(\rho_i)).S_i \,\&\, \mathbf{p} \,\&_{j\in J\setminus I}\, ?l_j(\exists[\rho_j|\{\rho_j\mapsto U_j\}].\mathtt{tr}(\rho_j)).S'_j$$

$$\mathbf{p} \oplus_{i\in I} !l_i(\exists[\rho_i|\{\rho_i\mapsto U_i\}].\mathtt{tr}(\rho_i)).S_i \sqcap \mathbf{p} \oplus_{i\in I} !l_i(\exists[\rho_i|\{\rho_i\mapsto U_i\}].\mathtt{tr}(\rho_i)).S_i \triangleq \mathbf{p} \oplus_{i\in I} !l_i(\exists[\rho_i|\{\rho_i\mapsto U_i\}].\mathtt{tr}(\rho_i)).S_i$$

Projecting a pack interaction between $\mathbf{p}$ and $\mathbf{p}'$ onto either $\mathbf{p}$ or $\mathbf{p}'$ produces a pack select or a pack branch. Projecting onto a different role $\mathbf{q}$ ignores the pack interaction and combines the projections of the continuations using the merge operator.

Definition 4.3.2. For a session type S, $\mathtt{roles}(S)$ denotes the set of roles occurring in S. We write $\mathbf{p} \in S$ for $\mathbf{p} \in \mathtt{roles}(S)$, and $\mathbf{p} \in S \setminus \mathbf{q}$ for $\mathbf{p} \in \mathtt{roles}(S) \setminus \{\mathbf{q}\}$.

Partial session types defined in Figure4.5 have the same cases as local types: terminated session, selection and branching, pack selection and pack branching, type variable and recursive type, but without role annotations. Similar to global and local types, partial types also have a notion of *duality*. As before the dual of branching is selection, and the dual of pack branching is pack selection, while the payload types are preserved. The duals for termination, type variable and recursive type are themselves.

Definition 4.3.3. $\overline{H}$ is the *dual of* H, defined by:

$$\overline{\oplus_{i\in I} !l_i(U_i).H_i} \triangleq \&_{i\in I}\, ?l_i(U_i).\overline{H_i} \qquad \overline{\&_{i\in I} ?l_i(U_i).H_i} \triangleq \oplus_{i\in I}\, !l_i(U_i).\overline{H_i}$$

$$\overline{\oplus_{i\in I} !l_i(\exists[\rho_i|\{\rho_i\mapsto U_i\}].\mathtt{tr}(\rho_i)).H_i} \triangleq \&_{i\in I}\, ?l_i(\exists[\rho_i|\{\rho_i\mapsto U_i\}].\mathtt{tr}(\rho_i)).\overline{H_i}$$

$$\overline{\&_{i\in I}\, ?l_i(\exists[\rho_i|\{\rho_i\mapsto U_i\}].\mathtt{tr}(\rho_i)).H_i} \triangleq \oplus_{i\in I}\, !l_i(\exists[\rho_i|\{\rho_i\mapsto U_i\}].\mathtt{tr}(\rho_i)).\overline{H_i}$$

$$\overline{\mathtt{end}} \triangleq \mathtt{end} \qquad \overline{\mathtt{t}} \triangleq \mathtt{t} \qquad \overline{\mu t.H} \triangleq \mu t.\overline{H}$$

Similar to the projection of global types to local types, a local type can be projected onto a role $\mathbf{q}$ giving a partial type. The resulting partial type describes only the communications in S that involve role $\mathbf{q}$. This definition follows the same principles as the previous Definition 4.3.1.

Definition 4.3.4. $S \upharpoonright \mathbf{q}$ is the *partial projection of* S *onto* $\mathbf{q}$:

$$\mathtt{end}\upharpoonright \mathbf{q} \triangleq \mathtt{end} \qquad \mathtt{t}\upharpoonright \mathbf{q} \triangleq \mathtt{t} \qquad (\mu \mathtt{t}.S)\upharpoonright \mathbf{q} \triangleq \begin{cases} \mu\, \mathtt{t}.(S\upharpoonright \mathbf{q}) & \text{if } S\upharpoonright \mathbf{q} \neq \mathtt{t}'\ (\forall \mathtt{t}') \\ \mathtt{end} & \text{otherwise} \end{cases}$$

$$(\mathbf{p} \oplus_{i\in I} !l_i(U_i).S_i)\upharpoonright \mathbf{q} \triangleq \begin{cases} \oplus_{i\in I}\, !l_i(U_i).(S_i\upharpoonright \mathbf{q}) & \text{if } \mathbf{q} = \mathbf{p}, \\ \bigsqcap_{i\in I}(S_i\upharpoonright \mathbf{q}) & \text{if } \mathbf{p} \neq \mathbf{q} \end{cases}$$

$$(\mathbf{p} \,\&_{i\in I} ?l_i(U_i).S_i)\upharpoonright \mathbf{q} \triangleq \begin{cases} \&_{i\in I}\, ?l_i(U_i).S_i\upharpoonright \mathbf{q} & \text{if } \mathbf{q} = \mathbf{p}, \\ \bigsqcap_{i\in I}(S_i\upharpoonright \mathbf{q}) & \text{if } \mathbf{p} \neq \mathbf{q} \end{cases}$$

$$(\mathbf{p} \oplus_{i\in I} !l_i(\exists[\rho_i|\{\rho_i\mapsto U_i\}].\mathtt{tr}(\rho_i)).S_i)\upharpoonright \mathbf{q} \triangleq \begin{cases} \oplus_{i\in I}\, !l_i(\exists[\rho_i|\{\rho_i\mapsto U_i\}].\mathtt{tr}(\rho_i)).(S_i\upharpoonright \mathbf{q}) & \text{if } \mathbf{q} = \mathbf{p}, \\ \bigsqcap_{i\in I}(S_i\upharpoonright \mathbf{q}) & \text{if } \mathbf{p} \neq \mathbf{q} \end{cases}$$

$$(\mathbf{p} \,\&_{i\in I}\, ?l_i(\exists[\rho_i|\{\rho_i\mapsto U_i\}].\mathtt{tr}(\rho_i)).S_i)\upharpoonright \mathbf{q} \triangleq \begin{cases} \&_{i\in I}\, ?l_i(\exists[\rho_i|\{\rho_i\mapsto U_i\}].\mathtt{tr}(\rho_i)).S_i\upharpoonright \mathbf{q} & \text{if } \mathbf{q} = \mathbf{p}, \\ \bigsqcap_{i\in I}(S_i\upharpoonright \mathbf{q}) & \text{if } \mathbf{p} \neq \mathbf{q} \end{cases}$$

Where the *merge operator for partial session types*, $\bigsqcap$, is defined by:

$$\mathtt{end}\sqcap\mathtt{end} \triangleq \mathtt{end} \qquad \mathtt{t}\sqcap\mathtt{t} \triangleq \mathtt{t} \qquad \mu t.H \sqcap \mu t.H' \triangleq \mu t.(H \sqcap H')$$

$$\&_{i\in I}\, ?l_i(U_i).H_i \sqcap \&_{i\in I}\, ?l_i(U_i).H_i' \triangleq \&_{i\in I}\, ?l_i(U_i).(H_i \sqcap H_i')$$

$$\oplus_{i\in I}\, !l_i(U_i).H_i \sqcap \oplus_{j\in J}\, !l_j(U_j).H_j' \triangleq \left(\oplus_{k\in I\cap J}\, !l_k(U_k).(H_k \sqcap H_k')\right) \oplus \left(\oplus_{i\in I\setminus J}\, !l_i(U_i).H_i\right) \oplus \left(\oplus_{j\in J\setminus I}\, !l_j(U_j).H_j'\right)$$

$$\&_{i\in I}\, ?l_i(\exists[\rho_i|\{\rho_i\mapsto U_i\}].\mathtt{tr}(\rho_i)).H_i \sqcap \&_{i\in I}\, ?l_i(\exists[\rho_i|\{\rho_i\mapsto U_i\}].\mathtt{tr}(\rho_i)).H_i' \triangleq \&_{i\in I}\, ?l_i(\exists[\rho_i|\{\rho_i\mapsto U_i\}].\mathtt{tr}(\rho_i)).(H_i \sqcap H_i')$$

$$\oplus_{i\in I}\, !l_i(\exists[\rho_i|\{\rho_i\mapsto U_i\}].\mathtt{tr}(\rho_i)).H_i \sqcap \oplus_{j\in J}\, !l_j(\exists[\rho_j|\{\rho_j\mapsto U_j\}].\mathtt{tr}(\rho_j)).H_j' \triangleq \left(\oplus_{k\in I\cap J}\, !l_k(\exists[\rho_k|\{\rho_k\mapsto U_k\}].\mathtt{tr}(\rho_k)).(H_k \sqcap H_k')\right) \oplus \left(\oplus_{i\in I\setminus J}\, !l_i(\exists[\rho_i|\{\rho_i\mapsto U_i\}].\mathtt{tr}(\rho_i)).H_i\right) \oplus \left(\oplus_{j\in J\setminus I}\, !l_j(\exists[\rho_j|\{\rho_j\mapsto U_j\}].\mathtt{tr}(\rho_j)).H_j'\right)$$

Example 2 (Projections of Global and Local Types)**.** *Consider the global type G of the producer-consumer example from the introduction.*

$$G = \mathbf{p}\to\mathbf{c}\colon \mathsf{buff}(\exists[\rho_\mathbf{q}|\{\rho_\mathbf{q}\mapsto S'_\mathbf{q}\}].\mathtt{tr}(\rho_\mathbf{q})).\mu\mathtt{t}.\mathbf{q}\to\mathbf{b}\colon \mathsf{add}(\mathtt{Int}).\mathbf{p}\to\mathbf{c}\colon \mathsf{turn}(\{\rho_\mathbf{q}\mapsto S'_\mathbf{q}\}).\ \mathbf{q}\to\mathbf{b}\colon \mathsf{req}(\mathtt{Str}).\mathbf{b}\to\mathbf{q}\colon \mathsf{snd}(\mathtt{Int}).\mathbf{c}\to\mathbf{p}\colon \mathsf{turn}(\{\rho_\mathbf{q}\mapsto S_\mathbf{q}\}).\mathtt{t}$$

It captures the interaction between the producer and consumer entities through roles **p**, **c**, *and between producer, consumer and buffer through roles* **q** *(shared between producer and consumer)*

$$[\text{SBR}]\ \frac{\forall i \in I \qquad U_i \leqslant U_i' \qquad S_i \leqslant S_i'}{\mathbf{p}\ \&_{i\in I} ?l_i(U_i).S_i \leqslant \mathbf{p}\ \&_{i\in I\cup J}\ ?l_i(U_i').S_i'}$$

$$[\text{SSEL}]\ \frac{\forall i \in I \qquad U_i' \leqslant U_i \qquad S_i \leqslant S_i'}{\mathbf{p}\ \oplus_{i\in I\cup J}\ !l_i(U_i).S_i \leqslant \mathbf{p}\ \oplus_{i\in I}\ !l_i(U_i').S_i'}$$

$$[\text{SBRP}]\ \frac{\forall i \in I \qquad U_i \leqslant U_i' \qquad S_i \leqslant S_i'}{\mathbf{p}\ \&_{i\in I}\ ?l_i(\exists[\rho_i|\{\rho_i \mapsto U_i\}].\mathtt{tr}(\rho_i)).S_i \leqslant \mathbf{p}\ \&_{i\in I\cup J}\ ?l_i(\exists[\rho_i|\{\rho_i \mapsto U_i'\}].\mathtt{tr}(\rho_i)).S_i'}$$

$$[\text{SSELP}]\ \frac{\forall i \in I \qquad U_i' \leqslant U_i \qquad S_i \leqslant S_i'}{\mathbf{p}\ \oplus_{i\in I\cup J}\ !l_i(\exists[\rho_i|\{\rho_i \mapsto U_i\}].\mathtt{tr}(\rho_i)).S_i \leqslant \mathbf{p}\ \oplus_{i\in I}\ !l_i(\exists[\rho_i|\{\rho_i \mapsto U_i'\}].\mathtt{tr}(\rho_i)).S_i'}$$

$$[\text{SB}]\ \frac{B \leqslant B'}{B \leqslant B'} \qquad [\text{SEND}]\ \frac{}{\mathtt{end} \leqslant \mathtt{end}} \qquad [\text{S}\mu\text{L}]\ \frac{S\{\mu \mathtt{t}.S/\mathtt{t}\} \leqslant S'}{\mu \mathtt{t}.S \leqslant S'} \qquad [\text{S}\mu\text{R}]\ \frac{S \leqslant S'\{\mu \mathtt{t}.S'/\mathtt{t}\}}{S \leqslant \mu \mathtt{t}.S'}$$

Figure 4.6: Subtyping for local session types.

and $\mathbf{b}$. *Projecting onto* $\mathbf{p}$ *gives the session type* z

$$S = G \upharpoonright \mathbf{p} = \mathbf{c}!\mathsf{buff}(\exists[\rho_\mathbf{q}|\{\rho_\mathbf{q} \mapsto S_\mathbf{q}'\}].\mathtt{tr}(\rho_\mathbf{q})).\mu \mathtt{t}.\mathbf{c}!\mathsf{turn}(\{\rho_\mathbf{q} \mapsto S_\mathbf{q}'\}).\mathbf{c}?\mathsf{turn}(\{\rho_\mathbf{q} \mapsto S_\mathbf{q}\}).\mathtt{t}$$

and further projecting onto $\mathtt{c}$ *gives the partial session type:*

$$H = S \upharpoonright \mathbf{c} = !\mathsf{buff}(\exists[\rho_\mathbf{q}|\{\rho_\mathbf{q} \mapsto S_\mathbf{q}'\}].\mathtt{tr}(\rho_\mathbf{q})).\mu\, \mathtt{t}.!\mathsf{turn}(\{\rho_\mathbf{q} \mapsto S_\mathbf{q}'\}).?\mathsf{turn}(\{\rho_\mathbf{q} \mapsto S_\mathbf{q}\}).\mathtt{t}$$

Definition 4.3.5 (Subtyping)**.** *Subtyping on session types* $\leqslant$ is the largest relation such that: *(i)* if $S \leqslant S'$, then $\forall \mathbf{p} \in (\mathtt{roles}(S) \cup \mathtt{roles}(S'))\ S \upharpoonright \mathbf{p} \leqslant S' \upharpoonright \mathbf{p}$, and *(ii)* is closed backwards under the coinductive rules in Figure4.6. *Subtyping on partial session types* $\leqslant$ is defined coinductively by the rules in Figure4.7.

Capabilities In our type system linearity is enforced via capabilities, rather than via environment splitting as in most session type systems. Each process has a capability set C associated with it, allowing it to communicate on the associated channels. The tracked type $\mathtt{tr}(\rho)$ is a singleton type associating a channel to capability ρ and to no other, which in turn maps to the channel's session type $\{\rho \mapsto S\}$. Hence two variables with the same capability ρ are aliases for the same channel. Individual capabilities are joined together using the $\otimes$ operator: $C = \{\rho_1 \mapsto S_1\} \otimes \ldots \otimes \{\rho_n \mapsto S_n\}$. The ordering is insignificant. The type system maintains the invariant that $\rho_1, \ldots, \rho_n$ are distinct.

Definition 4.3.6 (Terminated capabilities)**.** A capability set C is *terminated* if for every $\rho \in \mathrm{dom}(C)$, $C(\rho) = \mathtt{end}$.

$$[\text{SPARBR}]\ \frac{\forall i \in I \qquad U_i \leqslant U_i' \qquad H_i \leqslant H_i'}{\&_{i\in I}?l_i(U_i).H_i \leqslant \&_{i\in I\cup J}?l_i(U_i').H_i'}$$

$$[\text{SPARSEL}]\ \frac{\forall i \in I \qquad U_i' \leqslant U_i \qquad H_i \leqslant H_i'}{\oplus_{i\in I\cup J}!l_i(U_i).H_i \leqslant \oplus_{i\in I}!l_i(U_i').H_i'}$$

$$[\text{SPARBRP}]\ \frac{\forall i \in I \qquad U_i \leqslant U_i' \qquad H_i \leqslant H_i'}{\&_{i\in I}?l_i(\exists[\rho_i|\{\rho_i\mapsto U_i\}].\mathtt{tr}(\rho_i)).H \leqslant \&_{i\in I\cup J}?l_i(\exists[\rho_i|\{\rho_i\mapsto U_i'\}].\mathtt{tr}(\rho_i)).H_i'}$$

$$[\text{SPARSELP}]\ \frac{\forall i \in I \qquad U_i' \leqslant U_i \qquad H_i \leqslant H_i'}{\oplus_{i\in I\cup J}!l_i(\exists[\rho_i|\{\rho_i\mapsto U_i\}].\mathtt{tr}(\rho_i)).H_i \leqslant \oplus_{i\in I}!l_i(\exists[\rho_i|\{\rho_i\mapsto U_i'\}].\mathtt{tr}(\rho_i)).H_i'}$$

$$[\text{SPAREND}]\ \frac{}{\mathtt{end}\leqslant\mathtt{end}} \qquad [\text{SPAR}\mu\text{L}]\ \frac{H\{\mu\,\mathtt{t}.H/\mathtt{t}\}\leqslant H'}{\mu\,\mathtt{t}.H\leqslant H'} \qquad [\text{SPAR}\mu\text{R}]\ \frac{H\leqslant H'\{\mu\,\mathtt{t}.H'/\mathtt{t}\}}{H\leqslant\mu\,\mathtt{t}.H'}$$

Figure 4.7: Subtyping for partial session types.

Definition 4.3.7 (Substitution of capabilities)**.**

$$\begin{array}{rclcrcl}\{\rho\mapsto S\}[\rho'/\rho_2] & = & \{\rho\mapsto S\} & & \{\rho\mapsto S\}[\rho'/\rho] & = & \{\rho'\mapsto S\}\\ \varnothing[\rho'/\rho] & = & \varnothing & & (C_1\otimes C_2)[\rho'/\rho] & = & C_1[\rho'/\rho]\otimes C_2[\rho'/\rho]\end{array}$$

4.4 Typing

Definition 4.4.1. Typing judgements are inductively defined by the rules in Figure 4.8, and have the form: $\Gamma\vdash v : T;C$ for values, or $\Delta;\Gamma\vdash P;C$ for processes (with (Γ,C) consistent, and $\forall(\mathtt{c}:\mathtt{tr}(\rho)\in\Gamma;\{\rho\mapsto S\}\in C), S\upharpoonright\mathbf{p}$ is defined $\forall\mathbf{p}\in S$).

Γ is an environment of typed variables and channels together with their capability typing. Δ, defined in Figure 4.5 is an environment of typed process names, used in rules [TDEF] and [TCALL] for recursive process definitions and calls. If a channel $\mathtt{s}[\mathbf{p}]$ is in Γ, with type $\mathtt{tr}(\rho)$, then Γ also contains $\rho : \{\rho\mapsto S\}$ for some session type S. The capability ρ might, or might not, be in C, to show whether or not the channel can be used. If ρ is in C, then it occurs with the same session type: $\{\rho\mapsto S\}$. Rule [TCAP] takes the type for a capability ρ from the capability set. [TVAR] and [TVAL] are standard.

[TINACT] has a standard condition that all session types have reached `end`, expressed as the capability set being terminated. [TPAR] combines the capability sets in a parallel composition. [TSUB] is a standard subsumption rule using $\leqslant$ (Definition 4.3.5), the difference being the type in the capability set. [TSEL] states that selection on channel $\mathtt{c}[\mathbf{p}]$ is well typed if the capability associated with it is of compatible selection type and the continuation P is well-typed with the continuation session type. [TBR] states that branching on channel $\mathtt{c}[\mathbf{p}]$ is well typed if the

capability associated with it is of compatible branching type and the continuations $P_i, \forall i \in I$ are well-typed with the continuation session types. [TSELP] is similar to [TSEL], with the notable difference that an existential package is created for the channel being sent, containing the channel and its abstracted capability. Note that the actual capability to use the endpoint remains with process P. [TBRP] is similar to [TBR], with the difference that it unpackages the channel received and binds its capability type in the continuation session type (used to identify the correct capability when received later). [TRES] requires the restricted environment Γ' and the associated capability set C' to be *complete* (Definition 4.5.1). [TDEF] takes account of capability sets as well as parameters, and [TCALL] similarly requires capability sets. The parameters of a defined process include any necessary capabilities, which then also appear in the corresponding C_i, because not all capabilities associated with the channel parameters need to be present when the call is made.

4.5 Main Results

4.5.1 Subject Reduction

There are two important concepts relating the environment Γ and the capability set C: *completeness* and *consistency*, used in our type system.

Completeness means that if a channel is in Γ and its capability is in C, then Γ also contains the other endpoints of the channel and C contains the corresponding capability. In this case, there is a self-contained collection of channels that can be used to communicate. Consistency means that the opposite endpoints of every channel have dual partial types.

Definition 4.5.1 (Completeness and consistency)**.**

(Γ, C) is *complete* if and only if for all $\mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho_\mathbf{p})$ with $\rho_\mathbf{p} : \{\rho_\mathbf{p} \mapsto S_\mathbf{p}\} \in \Gamma$ and $\{\rho_\mathbf{p} \mapsto S_\mathbf{p}\} \in C$, $\mathbf{q} \in S_\mathbf{p}$ implies $\mathtt{s}[\mathbf{q}] : \mathtt{tr}(\rho_\mathbf{q}), \rho_\mathbf{q} : \{\rho_\mathbf{q} \mapsto S_\mathbf{q}\} \in \Gamma$ and $\{\rho_\mathbf{q} \mapsto S_\mathbf{q}\} \in C$.

(Γ, C) is *consistent* if and only if for all $\mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho_\mathbf{p}), \mathtt{s}[\mathbf{q}] : \mathtt{tr}(\rho_\mathbf{q}), \rho_\mathbf{p} : \{\rho_\mathbf{p} \mapsto S_\mathbf{p}\}, \rho_\mathbf{q} : \{\rho_\mathbf{q} \mapsto S_\mathbf{q}\} \in \Gamma$ we have $\overline{S_\mathbf{p} \upharpoonright \mathbf{q}} \leqslant S_\mathbf{q} \upharpoonright \mathbf{p}$.

Following standard practice in the multiparty session type literature, we show type safety and hence communication safety by proving subject reduction Theorem 4.5.2. In the usual way, session types evolve during reduction — in our system, this is seen in both the Γ environment and the capability set C.

Definition 4.5.2 (Typing context reduction)**.** The *reduction* $(\Gamma;\ C) \longrightarrow (\Gamma';\ C')$ is:

$$(\mathtt{s}[\mathbf{p}]:\mathtt{tr}(\rho_\mathbf{p}),\mathtt{s}[\mathbf{q}]:\mathtt{tr}(\rho_\mathbf{q}),\rho_\mathbf{p}:\{\rho_\mathbf{p}\mapsto S_\mathbf{p}\},\rho_\mathbf{q}:\{\rho_\mathbf{q}\mapsto S_\mathbf{q}\};\{\rho_\mathbf{p}\mapsto S_\mathbf{p},\rho_\mathbf{q}\mapsto S_\mathbf{q}\}) \longrightarrow$$
$$(\mathtt{s}[\mathbf{p}]:\mathtt{tr}(\rho_\mathbf{p}),\mathtt{s}[\mathbf{q}]:\mathtt{tr}(\rho_\mathbf{q}),\rho_\mathbf{p}:\{\rho_\mathbf{p}\mapsto S_k\},\rho_\mathbf{q}:\{\rho_\mathbf{q}\mapsto S'_k\};\{\rho_\mathbf{p}\mapsto S_k,\rho_\mathbf{q}\mapsto S'_k\})$$
$$\text{if } \begin{cases}\mathtt{unf}(S_\mathbf{p}) = \mathbf{q}\oplus_{i\in I}!l_i(U_i).S_i & k\in I\\ \mathtt{unf}(S_\mathbf{q}) = \mathbf{p}\,\&_{i\in I\cup J}\,?l_i(U'_i).S'_i & U_k\leqslant U'_k\end{cases}$$
$$\text{or if } \begin{cases}\mathtt{unf}(S_\mathbf{p}) = \mathbf{q}\oplus_{i\in I}!l_i(\exists[\rho_i|\{\rho_i\mapsto U_i\}].\mathtt{tr}(\rho_i)).S_i & k\in I\\ \mathtt{unf}(S_\mathbf{q}) = \mathbf{p}\,\&_{i\in I\cup J}\,?l_i(\exists[\rho_i|\{\rho_i\mapsto U'_i\}].\mathtt{tr}(\rho_i)).S'_i & U_k\leqslant U'_k\end{cases}$$
$$(\Gamma,\mathtt{c}:\mathtt{tr}(\rho),\rho:\{\rho\mapsto U\};\ C\otimes\{\rho\mapsto U\}) \longrightarrow (\Gamma',\mathtt{c}:\mathtt{tr}(\rho),\rho:\{\rho\mapsto U'\};\ C'\otimes\{\rho\mapsto U'\})$$
$$\text{if } (\Gamma;\ C)\longrightarrow(\Gamma';\ C') \text{ and } U\leqslant U'$$

Following [122] our Definition 4.5.2 also accommodates subtyping ($\leqslant$) and iso-recursive type equivalence (hence, unfolds types explicitly). We define $\longrightarrow^*$ as the reflexive and transitive closure of $\longrightarrow$.

Before giving the subject reduction theorem, we show several properties needed for proving the theorem.

Lemma 4.5.1. *If* $(\Gamma;\ C)\longrightarrow(\Gamma';\ C')$ *and* $(\Gamma;\ C)$ *is* consistent (resp. complete)*, then so is* $(\Gamma';\ C')$.

Proof Sketch. By induction on $(\Gamma;\ C)\longrightarrow(\Gamma';C')$, as per Definition 4.5.2. The full proof can be found in the appendix Lemma A.1. □

Corollary 4.5.1. *If* $(\Gamma_1,\Gamma_2;C_1\otimes C_2)$ *is consistent and* $(\Gamma_1;C_1)\longrightarrow^*(\Gamma'_1;C'_1)$*, then* $(\Gamma'_1,\Gamma_2;C'_1\otimes C_2)$ *is consistent.*

Proof Sketch. By induction on the reduction, and induction on the size of $(\Gamma_2;C_2)$. The full proof can be found in the appendix A.1. □

Proposition 4.5.1 (Weakening)**.** *For any multiparty session process* P *with* $\Delta;\Gamma\vdash P;\ C$*:*

1. *if* Δ *and* Δ' *are disjoint, then* $\Delta,\Delta';\Gamma\vdash P;\ C$.
2. *if* Γ *and* Γ' *are disjoint, then* $\Delta;\Gamma,\Gamma'\vdash P;\ C$.

Proof Sketch. By induction on typing derivations, with a case analysis on the last rule applied. The full proof can be found in the appendix Section A.1. □

Proposition 4.5.2 (Strengthening)**.** *For any multiparty session process* P*:*

1. *if* $\Delta,\Delta';\Gamma\vdash P;\ C$ *and* $\Delta'\notin fpv(P)$ *then* $\Delta;\Gamma\vdash P;\ C$.

2. *if* $\Delta;\Gamma,\Gamma' \vdash P;\ C$ *and* $\Gamma' \notin fv(P)$ *and* $\Gamma' \notin C$, *then* $\Delta;\Gamma \vdash P;\ C$.

Proof Sketch. By induction on typing derivations, with a case analysis on the last rule applied. The full proof can be found in the appendix A.1. □

Proposition 4.5.3. *For all multiparty session processes* P, P' *if* $\Delta;\Gamma \vdash P;\ C$ *and* $P \equiv P'$, *then* $\Delta;\Gamma \vdash P';\ C$.

Proof Sketch. By induction on the structural congruence. The full proof can be found in the appendix A.1. □

Definition 4.5.3 (Substitution of Capabilities)**.** Is defined as:

$$\{\rho \mapsto S\}[\rho'/\rho_2] = \{\rho \mapsto S\} \qquad \{\rho \mapsto S\}[\rho'/\rho] = \{\rho' \mapsto S\}$$
$$\varnothing[\rho'/\rho] = \varnothing \qquad (C_1 \otimes C_2)[\rho'/\rho] = C_1[\rho'/\rho] \otimes C_2[\rho'/\rho]$$

Lemma 4.5.1 (Substitution)**.** 1. *If* $\Delta;\Gamma, \mathtt{x} : U \vdash P;\ C$ *and* $\Gamma' \vdash \mathtt{v} : U;\varnothing$ *then* $\Delta;\Gamma,\Gamma' \vdash P\{v/x\};C$.

2. *If* $\Delta;\Gamma,\rho : \{\rho \mapsto S\} \vdash P;\ C \otimes \{\rho \mapsto S\},\ \Gamma' \vdash \rho' : \{\rho' \mapsto S\};\{\rho' \mapsto S\}$ *and* $(\Gamma,\Gamma';\ C \otimes \{\rho' \mapsto S\})$ *consistent, then* $\Delta;\Gamma,\Gamma' \vdash P\{\rho'/\rho\};\ C \otimes \{\rho' \mapsto S\}$.

3. *If* $\Delta;\Gamma, \mathbf{c} : \mathtt{tr}(\rho), \rho : \{\rho \mapsto S\} \vdash P;\ C$, *and* $\mathbf{c}' : \mathtt{tr}(\rho'), \rho' : \{\rho' \mapsto S\} \in \Gamma';\{\rho' \mapsto S\}$ *and* $(\Gamma,\Gamma';\ C)$ *consistent, then* $\Delta;\Gamma,\Gamma' \vdash P\{\rho'/\rho\};\ C$.

Proof Sketch. By induction on the derivation of $\Delta;\Gamma, x : U \vdash P;\ C$ and $\Delta;\Gamma,\rho : \{\rho \mapsto S\} \vdash P;\ C \otimes \{\rho \mapsto S\}$, with a case analysis on the last rule applied. The full proof can be found in the appendix A.1. □

Definition 4.5.4. For all pairs of multiparty session typing contexts and capability collections $(\Gamma;\ C)$, $(\Gamma';\ C')$, the relation $C \leqslant C'$ holds if and only if $\mathrm{dom}(\Gamma) = \mathrm{dom}(\Gamma')$, $\mathrm{dom}(C) = \mathrm{dom}(C')$ and $\forall \mathbf{c} : \mathtt{tr}(\rho), \rho : \{\rho \mapsto U\} \in \mathrm{dom}(\Gamma)$ then $C(\rho) \leqslant C'(\rho)$. The following rule is defined, corresponding to 0 or more consecutive applications of [TSUB]:

$$\frac{\Delta;\Gamma \vdash P;\ C \qquad C' \leqslant C}{\Delta;\Gamma \vdash P;\ C_1} \text{[TMSUB]}$$

Proposition 4.5.1. *If* $\Delta;\Gamma \vdash P \,|\, Q;\ C$ *then there exist* C_1, C_2, C_1', C_2' *such that* $C = C_1 \otimes C_2$, $C_1 \leqslant C_1'$,

$C_2 \leqslant C_2'$, $\Delta;\Gamma \vdash P;\ C_1'$, $\Delta;\Gamma \vdash Q;\ C_2'$. *Moreover:*

$$\dfrac{\dfrac{\dfrac{\Delta;\Gamma\vdash P;\ C_1' \qquad C_1 \leqslant C_1'}{\Delta;\Gamma\vdash P;\ C_1}\,[\text{TMSUB}] \qquad \dfrac{\Delta;\Gamma\vdash Q;\ C_2' \qquad C_2 \leqslant C_2'}{\Delta;\Gamma\vdash Q;\ C_2}\,[\text{TMSUB}]}{\Delta;\Gamma\vdash P\,|\,Q;\ C_1\otimes C_2}\,[\text{TPAR}]$$

if and only if

$$\dfrac{\dfrac{C_1\otimes C_2 \leqslant C_1'\otimes C_2' \qquad \dfrac{\Delta;\Gamma\vdash P;\ C_1' \qquad \Delta;\Gamma\vdash Q;\ C_2'}{\Delta;\Gamma\vdash P\,|\,Q;\ C_1'\otimes C_2'}\,[\text{TPAR}]}{\Delta;\Gamma\vdash P\,|\,Q;\ C_1\otimes C_2}\,[\text{TMSUB}]$$

if and only if

$$\dfrac{\dfrac{\dfrac{\Delta;\Gamma\vdash P;\ C_1' \qquad \dfrac{\Delta;\Gamma\vdash Q;\ C_2' \qquad C_2 \leqslant C_2'}{\Delta;\Gamma\vdash Q;\ C_2}\,[\text{TMSUB}]}{\Delta;\Gamma\vdash P\,|\,Q;\ C_1'\otimes C_2}\,[\text{TPAR}] \qquad C_1\otimes C_2 \leqslant C_1'\otimes C_2}{\Delta;\Gamma\vdash P\,|\,Q;\ C_1\otimes C_2}\,[\text{TMSUB}]$$

if and only if

$$\dfrac{\dfrac{\dfrac{\Delta;\Gamma\vdash P;\ C_1' \qquad C_1 \leqslant C_1'}{\Delta;\Gamma\vdash P;\ C_1}\,[\text{TMSUB}] \qquad \Delta;\Gamma\vdash Q;\ C_2'}{\Delta;\Gamma\vdash P\,|\,Q;\ C_1\otimes C_2'}\,[\text{TPAR}] \qquad C_1\otimes C_2 \leqslant C_1\otimes C_2'}{\Delta;\Gamma\vdash P\,|\,Q;\ C_1\otimes C_2}\,[\text{TMSUB}]$$

Proof. The first part of the statement is proven by inversion of [TPAR], by Definition 4.5.4 and adding a possibly empty instance of [TMSUB], as in the last case in the "moreover" statement. The if and only if relations among the typing derivations are straightforward: the hypotheses of one derivation imply all the others, and if one hypothesis is falsified, none of the derivations hold. □

Proposition 4.5.2. *If* $\Delta;\Gamma_1 \vdash (\nu\, s : \Gamma_2)P;\ C_1$ *then there exist* $\Gamma_1', \Gamma_2', C_1', C_2'$ *such that:*

$(\Gamma_1; C_1) \leqslant (\Gamma_1'; C_1')$, $(\Gamma_2; C_2) \leqslant \Gamma_2'; C_2'$ *and* $\Delta; \Gamma_1, \Gamma_2 \vdash P; C_1' \otimes C_2'$. *Moreover:*

$$\dfrac{\dfrac{\Delta; \Gamma_1, \Gamma_2 \vdash P; C_1' \otimes C_2' \quad C_1 \otimes C_2 \leqslant C_1' \otimes C_2'}{\Delta; \Gamma_1, \Gamma_2 \vdash P; C_1 \otimes C_2}\,[\text{TMSUB}]}{\Delta; \Gamma_1 \vdash (\nu\, s : \Gamma_2)P; C_1}\,[\text{TRES}]$$

if and only if

$$\dfrac{\dfrac{\dfrac{\Delta; \Gamma_1, \Gamma_2 \vdash P; C_1' \otimes C_2' \qquad C_2 \leqslant C_2'}{\Delta; \Gamma_1, \Gamma_2 \vdash P; C_1' \otimes C_2}\,[\text{TMSUB}]}{\Delta; \Gamma_1 \vdash (\nu\, s : \Gamma_2)P; C_1'}\,[\text{TRES}] \qquad C_1 \leqslant C_1'}{\Delta; \Gamma_1 \vdash (\nu\, s : \Gamma_2)P; C_1}\,[\text{TMSUB}]$$

Proof. The first part of the statement is by inversion of [TRES], by Definition 4.5.4 and adding a possibly empty instance of [TMSUB], like in the first case in the "moreover" statement. The if and only if relations among the typing derivations are straightforward: the hypotheses of one derivation implies the other, and when one hypothesis is falsified, none of the derivations hold. □

Proposition 4.5.3 (Subtyping normalisation)**.** *If* $\Delta; \Gamma \vdash P; C$ *then there exists a derivation that proves the judgement by only applying rule* [TMSUB] *on the conclusions of* [TBR], [TSEL], [TBRP], [TSELP], *and* [TCALL].

Proof Sketch. Assume that we have a derivation D concluding $\Delta, \Gamma \vdash P; C$, that does not match the thesis: if it is just an instance of [TBR], [TSEL], [TBRP], [TSELP], or [TCALL], we conclude by adding a empty instance of [TMSUB]. Otherwise, D must have one of the shapes in Proposition 4.5.1 or Proposition 4.5.2, and we can apply [TMSUB] towards the leafs of the derivation tree, where [TBR], [TSEL], [TBRP], [TSELP], and [TCALL] occur, by recursively rewriting it in the first form of the statements. □

Theorem 4.5.2 (Subject reduction)**.** *If* $\Delta; \Gamma \vdash P; C$ *and* $P \longrightarrow P'$, *then there exist* Γ' *and* C' *such that* $\Delta; \Gamma' \vdash P'; C'$ *and* $(\Gamma; C) \longrightarrow^* (\Gamma'; C')$.

Proof. By induction on the reduction of $P \longrightarrow P'$, with an analysis of the derivation of $\Delta; \Gamma \vdash P; C$.

- **case [RCOM]**
  We have: $P = \mathtt{s}[\mathbf{p}][\mathbf{q}]\&_{i\in I}\{l_i(x_i).Q_i\} \mid \mathtt{s}[\mathbf{q}][\mathbf{p}] \oplus \langle l_j(v)\rangle.Q \longrightarrow Q_j\{v/x_j\} \mid Q = P'$, where if $j \in I$ and $fv(v) = \varnothing$, and $C = C_1 \otimes \{\rho_{\mathbf{p}} \mapsto \mathbf{p}\ \&_{i\in I}?l_i(U_i).S_i\} \otimes C_2 \otimes \{\rho_{\mathbf{q}} \mapsto \mathbf{q} \oplus_{j\in I} !l_j(U).S\}$.

  We distinguish two cases based on whether the payload communicated has a capability attached or not.

In the first case, the payload has no capability attached. We provide the derivation for P, allowing (possibly empty) instances of [TMSUB] as per Proposition 4.5.3:

$$\dfrac{C_1 \leqslant C_1' \quad \dfrac{\Delta;\Gamma, x_i : U_i' \vdash Q_i;\ C_1'^{\diamond} \otimes \{\rho_{\mathbf{p}} \mapsto S_i'\} \quad \mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho_{\mathbf{p}}), \rho_{\mathbf{p}} : \{\rho_{\mathbf{p}} \mapsto S_i'\} \in \Gamma}{\Delta;\Gamma \vdash \mathtt{s}[\mathbf{p}][\mathbf{q}] \&_{i\in I}\{l_i(x_i).Q_i\};\ C_1'^{\diamond} \otimes \{\rho_{\mathbf{p}} \mapsto \mathbf{p}\ \&_{i\in I}?l_i\,(U_i')\,.S_i'\}}\,[\text{TBR}]}{\Delta;\Gamma \vdash \mathtt{s}[\mathbf{p}][\mathbf{q}] \&_{i\in I}\{l_i(x_i).Q_i\};\ C_1 \otimes \{\rho_{\mathbf{p}} \mapsto \mathbf{p}\ \&_{i\in I}?l_i\,(U_i)\,.S_i\}}\,[\text{TMSUB}]$$

$$\vdots$$

$$\dfrac{C_2 \leqslant C_2' \quad \dfrac{\mathtt{s}[\mathbf{q}] : \mathtt{tr}(\rho_{\mathbf{q}}), \rho_{\mathbf{q}} : \{\rho_{\mathbf{q}} \mapsto S'\} \in \Gamma \quad \Delta;\Gamma \vdash Q;\ C_2'^{\diamond} \otimes \{\rho_{\mathbf{q}} \mapsto S'\} \quad \dfrac{v \in U'}{\Gamma \vdash v : U';\varnothing}\,[\text{TVAL}]}{\Delta;\Gamma \vdash \mathtt{s}[\mathbf{q}][\mathbf{p}] \oplus \langle l_j(v)\rangle.Q;\ C_2'^{\diamond} \otimes \{\rho_{\mathbf{q}} \mapsto \mathbf{q} \oplus_{j\in I} !l_j(U').S'\}}\,[\text{TSEL}]}{\Delta;\Gamma \vdash \mathtt{s}[\mathbf{q}][\mathbf{p}] \oplus \langle l_j(v)\rangle.Q;\ C_2}\,[\text{TMSUB}]$$

$$\dfrac{}{\Delta;\Gamma \vdash \mathtt{s}[\mathbf{p}][\mathbf{q}] \&_{i\in I}\{l_i(x_i).Q_i\} \mid \mathtt{s}[\mathbf{q}][\mathbf{p}] \oplus \langle l_j(v)\rangle.Q;\ C}\,[\text{TPAR}] \tag{4.1}$$

$$C = C_1 \otimes C_2 \tag{4.2}$$

$$C_1 = C_1^{\diamond} \otimes \{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}\}\ \text{, where } S_{\mathbf{p}} = \mathbf{p}\ \&_{i\in I}?l_i\,(U_i)\,.S_i \tag{4.3}$$

$$C_2 = C_2^{\diamond} \otimes \{\rho_{\mathbf{q}} \mapsto S_{\mathbf{q}}\}\ \text{, where } S_{\mathbf{q}} = \mathbf{q} \oplus_{j\in I} !l_j(U).S \tag{4.4}$$

$$C_1' = C_1'^{\diamond} \otimes \{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}'\}\ \text{, where } S_{\mathbf{p}}' = \mathbf{p}\ \&_{i\in I}?l_i\,(U_i')\,.S_i' \tag{4.5}$$

$$C_2' = C_2'^{\diamond} \otimes \{\rho_{\mathbf{q}} \mapsto S_{\mathbf{q}}'\}\ \text{, where } S_{\mathbf{q}}' = \mathbf{q} \oplus_{j\in I} !l_j(U').S' \tag{4.6}$$

From the consistency of $(\Gamma;\ C) = (\Gamma;\ C_1 \otimes C_2)$, and since $C_1 \leqslant C_1'$ and $C_2 \leqslant C_2'$ we also have:

$$U' \leqslant U_i' \tag{4.7}$$

Before proceeding, we prove $(\Gamma;\ C) \longrightarrow (\Gamma;\ C')$, and therefore, $(\Gamma;\ C) \longrightarrow^* (\Gamma;\ C')$:

We observe that:

$$(\mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho_{\mathbf{p}}), \mathtt{s}[\mathbf{q}] : \mathtt{tr}(\rho_{\mathbf{q}}); \{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}, \rho_{\mathbf{q}} \mapsto S_{\mathbf{q}}\}) \longrightarrow (\mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho_{\mathbf{p}}), \mathtt{s}[\mathbf{q}] : \mathtt{tr}(\rho_{\mathbf{q}}); \{\rho_{\mathbf{p}} \mapsto S_j', \rho_{\mathbf{q}} \mapsto S'\}) \tag{4.8}$$

since $(\Gamma;C)$ is consistent by hypothesis, and therefore $\mathtt{unf}(S_{\mathbf{p}} \upharpoonright \mathbf{q})$ and $\mathtt{unf}(S_{\mathbf{q}} \upharpoonright \mathbf{p})$ have at least l_j in common, with compatible payload types as per Definition 4.5.2.

Then we can prove the following statement:

$$(\Gamma;\ C_1^{\diamond} \otimes C_2^{\diamond} \otimes \{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}, \rho_{\mathbf{q}} \mapsto S_{\mathbf{q}}\}) \longrightarrow (\Gamma;\ C_1'^{\diamond} \otimes C_2'^{\diamond} \otimes \{\rho_{\mathbf{p}} \mapsto S_j', \rho_{\mathbf{q}} \mapsto S'\}) \tag{4.9}$$

by induction on the size of $C_1^{\diamond} \otimes C_2^{\diamond} \otimes \{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}, \rho_{\mathbf{q}} \mapsto S_{\mathbf{q}}\}$. The base case is given above, while for the inductive case we apply the induction hypothesis use the subtyping relations between the capability sets to conclude by the inductive rule of Definition 4.5.2.

We can now continue proving the main statement, observing:

$$(\Gamma;\ C') \text{ is consistent} \qquad \text{by (4.9) and Lemma 4.5.1} \qquad (4.10)$$

$$\Delta;\Gamma, x_j : U'_j \vdash Q_j;\ C'^{\diamond}_1 \otimes \{\rho_{\mathbf{p}} \mapsto S'_j\}\ (j \in I) \qquad \text{from (4.1), premise of [TBR]} \qquad (4.11)$$

$$\Delta;\Gamma, x_j : U' \vdash Q_j;\ C'^{\diamond}_1 \otimes \{\rho_{\mathbf{p}} \mapsto S'_j\} \qquad \text{from (4.11), (4.7), and [TSUB]} \qquad (4.12)$$

$$\Gamma \vdash v : U' \qquad \text{from (4.1), premise of [TSEL]} \qquad (4.13)$$

$$(\Gamma;\ C'^{\diamond}_1 \otimes \{\rho_{\mathbf{p}} \mapsto S'_j\}) \text{ is consistent} \qquad \text{from the above} \qquad (4.14)$$

$$\Delta;\Gamma \vdash Q_j\{v/x_j\};\ C'^{\diamond}_1 \otimes \{\rho_{\mathbf{p}} \mapsto S'_j\} \qquad \text{from the above and Lemma 4.5.1} \qquad (4.15)$$

Therefore we conclude by:

$$\frac{\Delta;\Gamma \vdash Q_j\{v/x_j\};\ C'^{\diamond}_1 \otimes \{\rho_{\mathbf{p}} \mapsto S'_j\} \qquad \Delta;\Gamma \vdash Q;\ C'^{\diamond}_2 \otimes \{\rho_{\mathbf{q}} \mapsto S'\}}{\Delta;\Gamma \vdash Q_j\{v/x_j\} \mid Q;\ C'}\ [\text{TPAR}]$$

Second case, the payload has a capability attached, which means that the payload in itself must be a capability.

By inversion of [TPAR] and [TBR]/[TSEL], allowing (possibly empty) instances of [TMSUB] as per Proposition 4.5.3, there exist C_1, C_2 such that $C = C_1 \otimes C_2$, and C'_1, C'_2 such that:

$$\dfrac{\dfrac{\dfrac{\Delta;\Gamma, \rho_i : \{\rho_i \mapsto U'_i\} \vdash Q_i;\ C'^{\diamond}_1 \otimes \{\rho_{\mathbf{p}} \mapsto S'_i, \rho_i \mapsto U'_i\} \quad \mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho_{\mathbf{p}}), \rho_{\mathbf{p}} : \{\rho_{\mathbf{p}} \mapsto S'_i\} \in \Gamma}{\Delta;\Gamma \vdash \mathtt{s}[\mathbf{p}][\mathbf{q}] \&_{i\in I}\{l_i(\rho_i).Q_i\};\ C'^{\diamond}_1 \otimes \{\rho_{\mathbf{p}} \mapsto \mathbf{p}\ \&_{i\in I}?l_i(\{\rho_i \mapsto U'_i\}).S'_i\} \qquad C_1 \leqslant C'_1}\ [\text{TMSUB}]}{\Delta;\Gamma \vdash \mathtt{s}[\mathbf{p}][\mathbf{q}] \&_{i\in I}\{l_i(\rho_i).Q_i\};\ C_1} \qquad \vdots \qquad \dfrac{\dfrac{\dfrac{\mathtt{s}[\mathbf{q}] : \mathtt{tr}(\rho_{\mathbf{q}}), \rho_{\mathbf{q}} : \{\rho_{\mathbf{q}} \mapsto S'\} \in \Gamma}{\Gamma \vdash \rho : \{\rho \mapsto U'\}; \{\rho \mapsto U'\}}\ [\text{TCAP}] \qquad \Delta;\Gamma \vdash Q;\ C'^{\diamond}_2 \otimes \{\rho_{\mathbf{q}} \mapsto S'\}}{\Delta;\Gamma \vdash \mathtt{s}[\mathbf{q}][\mathbf{p}] \oplus \langle l_j(v)\rangle.Q;\ C'^{\diamond}_2 \otimes \{\rho_{\mathbf{q}} \mapsto \mathbf{q} \oplus_{j\in I} !l_j(\{\rho \mapsto U'\}).S'\} \otimes \{\rho \mapsto U'\}}\ [\text{TSEL}] \qquad \vdots \qquad C_2 \leqslant C'_2}{\Delta;\Gamma \vdash \mathtt{s}[\mathbf{q}][\mathbf{p}] \oplus \langle l_j(v)\rangle.Q; C_2}\ [\text{TMSUB}]}{\Delta;\Gamma \vdash \mathtt{s}[\mathbf{p}][\mathbf{q}] \&_{i\in I}\{l_i(\rho_i).Q_i\} \mid \mathtt{s}[\mathbf{q}][\mathbf{p}] \oplus \langle l_j(v)\rangle.Q; C}\ [\text{TPAR}] \qquad (4.16)$$

$$C = C_1 \otimes C_2 \tag{4.17}$$
$$C_1 = C_1^{\diamond} \otimes \{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}\} \text{ where } S_{\mathbf{p}} = \mathbf{p}\,\&_{i\in I}?l_i(U_i).S_i \tag{4.18}$$
$$C_2 = C_2^{\diamond} \otimes \{\rho_{\mathbf{q}} \mapsto S_{\mathbf{q}}\} \otimes \{\rho \mapsto U\} \text{ where } S_{\mathbf{q}} = \mathbf{q} \oplus_{j\in I} !l_j(U).S \tag{4.19}$$
$$C_1' = C_1'^{\diamond} \otimes \{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}'\} \text{ where } S_{\mathbf{p}}' = \mathbf{p}\,\&_{i\in I}?l_i(U_i').S_i' \tag{4.20}$$
$$C_2' = C_2'^{\diamond} \otimes \{\rho_{\mathbf{q}} \mapsto S_{\mathbf{q}}'\} \otimes \{\rho \mapsto U'\} \text{ where } S_{\mathbf{q}}' = \mathbf{q} \oplus_{j\in I} !l_j(U').S' \tag{4.21}$$

From the consistency of $(\Gamma;\, C) = (\Gamma;\, C_1 \otimes C_2)$, and since $C_1 \leqslant C_1'$ and $C_2 \leqslant C_2'$ we also have: $U' \leqslant U_i'$.

Before proceeding, we prove $(\Gamma;\, C) \longrightarrow (\Gamma;\, C')$, and therefore, $(\Gamma;\, C) \longrightarrow^* (\Gamma;\, C')$:

We observe that:

$$(\mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho_{\mathbf{p}}), \mathtt{s}[\mathbf{q}] : \mathtt{tr}(\rho_{\mathbf{q}}); \{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}, \rho_{\mathbf{q}} \mapsto S_{\mathbf{q}}\}) \longrightarrow (\mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho_{\mathbf{p}}), \mathtt{s}[\mathbf{q}] : \mathtt{tr}(\rho_{\mathbf{q}}); \{\rho_{\mathbf{p}} \mapsto S_j', \rho_{\mathbf{q}} \mapsto S'\}) \tag{4.22}$$

since $(\Gamma;\, C)$ is consistent by hypothesis, and therefore $\mathtt{unf}(S_{\mathbf{p}} \upharpoonright \mathbf{q})$ and $\mathtt{unf}(S_{\mathbf{q}} \upharpoonright \mathbf{p})$ have at least l_j in common, with compatible payload types as per Definition 4.5.2.

Then we can prove the following statement:

$$(\Gamma;\, C_1^{\diamond} \otimes C_2^{\diamond} \otimes \{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}, \rho_{\mathbf{q}} \mapsto S_{\mathbf{q}}, \rho \mapsto U\}) \longrightarrow (\Gamma;\, C_1'^{\diamond} \otimes C_2'^{\diamond} \otimes \{\rho_{\mathbf{p}} \mapsto S_j', \rho_{\mathbf{q}} \mapsto S', \rho \mapsto U'\}) \tag{4.23}$$

by induction on the size of $C_1 \otimes C_2$. The base case is given above, while for the inductive case we apply the induction hypothesis use the subtyping relations between the capability sets to conclude by the inductive rule of Definition 4.5.2.

We can now continue proving the main statement, observing:

$(\Gamma; C')$ is consistent	by 4.5.1	(4.24)
$\Delta; \Gamma, \rho_j : \{\rho_j \mapsto U_j'\} \vdash Q_j; C_1'^{\diamond} \otimes \{\rho_{\mathbf{p}} \mapsto S_j', \rho_j \mapsto U_j'\}$ $(j \in I)$	premise of [TBR]	(4.25)
$\Delta; \Gamma, \rho_j : \{\rho_j \mapsto U_j'\} \vdash Q_j; C_1'^{\diamond} \otimes \{\rho_{\mathbf{p}} \mapsto S_j', \rho_j \mapsto U'\}$ $(j \in I)$	premise of [TSUB]	(4.26)
$\Gamma \vdash \rho : \{\rho \mapsto U'\}; \{\rho \mapsto U'\}$	premise of [TSEL]	(4.27)
$(\Gamma, C_1'^{\diamond} \otimes \{\rho_{\mathbf{p}} \mapsto S_j', \rho \mapsto U'\})$ is consistent	the above and 4.5.1	(4.28)
$\Delta; \Gamma \vdash Q_j\{v/\rho_j\}; C_1'^{\diamond} \otimes \{\rho_{\mathbf{p}} \mapsto S_j', \rho \mapsto U'\}$	the above and 4.5.1	(4.29)

Therefore we conclude by:

$$\frac{\Delta; \Gamma \vdash Q_j\{v/\rho_j\}; C_1'^{\diamond} \otimes \{\rho_{\mathbf{p}} \mapsto S_j', \rho \mapsto U'\} \qquad \Delta; \Gamma \vdash Q;\, C_2'^{\diamond} \otimes \{\rho_{\mathbf{q}} \mapsto S'\}}{\Delta; \Gamma \vdash Q_j\{v/x_j\} \mid Q;\, C'} \text{ [TPAR]}$$

- **case [RCOMP]**

We have: $P = \mathtt{s}[\mathbf{p}][\mathbf{q}]\&_{i\in I}\{l_i(\mathsf{pack}(\rho_i,\mathtt{v}_i)).Q_i\} \mid \mathtt{s}[\mathbf{q}][\mathbf{p}]\oplus\langle l_j(\mathsf{pack}(\rho,\mathtt{v}))\rangle.Q \longrightarrow Q_j\{\mathtt{v}/\mathtt{v}_j\} \mid Q = P'$, where: if $j \in I$ and $fv(v) = \varnothing$.

$$\dfrac{\dfrac{\dfrac{\Delta;\Gamma,v_i:\mathtt{tr}(\rho_i),\rho_i:\{\rho_i\mapsto U_i'\}\vdash Q_i;\ C_1'^{\diamond}\otimes\{\rho_{\mathbf{p}}\mapsto S_i'\} \quad \mathtt{s}[\mathbf{p}]\in\Gamma}{\Delta;\Gamma\vdash \mathtt{s}[\mathbf{p}][\mathbf{q}]\ \&_{i\in I}\ ?l_i(\mathsf{pack}(\rho_i,\mathtt{v}_i)).Q_i;\ C_1'^{\diamond}\otimes\{\rho_{\mathbf{p}}\mapsto S_{\mathbf{p}}\}}\ [\text{TBRP}] \qquad C_1 \leqslant C_1'}{\Delta;\Gamma\vdash \mathtt{s}[\mathbf{p}][\mathbf{q}]\ \&_{i\in I}\ ?l_i(\mathsf{pack}(\rho_i,\mathtt{v}_i)).Q_i;\ C_1}\ [\text{TMSUB}]$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{v:\mathtt{tr}(\rho),\rho:\{\rho\mapsto U\}\in\Gamma}\ [\text{TVAR}]}{\Gamma\vdash v:\mathtt{tr}(\rho);\varnothing} \qquad \mathtt{s}[\mathbf{q}]:\mathtt{tr}(\rho_{\mathbf{q}})\in\Gamma \qquad j\in I}{\Delta;\Gamma\vdash Q;\ C_2'^{\diamond}\otimes\{\rho_{\mathbf{q}}\mapsto S'\}\otimes\{\rho\mapsto U'\}}\ [\text{TSELP}]}{\Delta;\Gamma\vdash \mathtt{s}[\mathbf{q}][\mathbf{p}]\oplus\langle l_j(\mathsf{pack}(\rho,\mathtt{v}))\rangle.Q;\ C_2'^{\diamond}\otimes\{\rho_{\mathbf{q}}\mapsto S_{\mathbf{q}}\}\otimes\{\rho\mapsto U'\}} \qquad C_2\leqslant C_2'}\ [\text{TMSUB}]$$

$$\dfrac{\Delta;\Gamma\vdash \mathtt{s}[\mathbf{p}][\mathbf{q}]\ \&_{i\in I}\ ?l_i(\mathsf{pack}(\rho_i,\mathtt{v}_i)).Q_i \mid \mathtt{s}[\mathbf{q}][\mathbf{p}]\oplus\langle l_j(\mathsf{pack}(\rho,\mathtt{v}))\rangle.Q;\ C_1\otimes C_2}{}\ [\text{TPAR}]$$

$$\begin{array}{lcl}
C & = & C_1\otimes C_2\\
C_1 & = & C_1^{\diamond}\otimes\{\rho_{\mathbf{p}}\mapsto S_{\mathbf{p}}\} \text{ where } S_{\mathbf{p}} = \mathbf{p}\ \&_{i\in I}?l_i\,(\exists[\rho_i|\{\rho_i\mapsto U_i\}].\mathtt{tr}(\rho_i))\,.S_i\\
C_2 & = & C_2^{\diamond}\otimes\{\rho\mapsto U\}\otimes\{\rho_{\mathbf{q}}\mapsto S_{\mathbf{q}}\} \text{ where } S_{\mathbf{q}} = \mathbf{q}\ \oplus_{j\in I}!l_j(\exists[\rho|\{\rho\mapsto U\}].\mathtt{tr}(\rho)).S\\
C_1' & = & C_1'^{\diamond}\otimes\{\rho_{\mathbf{p}}\mapsto S_{\mathbf{p}}'\} \text{ where } S_{\mathbf{p}}' = \mathbf{p}\ \&_{i\in I}?l_i\,(U_i')\,.S_i'\\
C_2' & = & C_2'^{\diamond}\otimes\{\rho\mapsto U'\}\otimes\{\rho_{\mathbf{q}}\mapsto S_{\mathbf{q}}'\} \text{ where } S_{\mathbf{q}}' = \mathbf{q}\ \oplus_{j\in I}!l_j(\exists[\rho|\{\rho\mapsto U'\}].\mathtt{tr}(\rho)).S'
\end{array}$$

From the consistency of $(\Gamma;\ C) = (\Gamma;\ C_1\otimes C_2)$, and since $C_1 \leqslant C_1'$ and $C_2 \leqslant C_2'$ we also have: $U \leqslant U_j'$.

Before proceeding, we prove $(\Gamma;\ C) \longrightarrow (\Gamma;\ C')$, and therefore, $(\Gamma;\ C) \longrightarrow^* (\Gamma;\ C')$:

We observe that:

$$(\mathtt{s}[\mathbf{p}]:\mathtt{tr}(\rho_{\mathbf{p}}),\mathtt{s}[\mathbf{q}]:\mathtt{tr}(\rho_{\mathbf{q}});\{\rho_{\mathbf{p}}\mapsto S_{\mathbf{p}},\rho_{\mathbf{q}}\mapsto S_{\mathbf{q}}\}) \longrightarrow (\mathtt{s}[\mathbf{p}]:\mathtt{tr}(\rho_{\mathbf{p}}),\mathtt{s}[\mathbf{q}]:\mathtt{tr}(\rho_{\mathbf{q}});\{\rho_{\mathbf{p}}\mapsto S_j',\rho_{\mathbf{q}}\mapsto S'\})$$

since $(\Gamma;\ C)$ is consistent by hypothesis, and therefore $\mathtt{unf}(S_{\mathbf{p}}\upharpoonright\mathbf{q})$ and $\mathtt{unf}(S_{\mathbf{q}}\upharpoonright\mathbf{p})$ have at least l_j in common, with compatible payload types as per Definition 4.5.2.

Then we can prove the following statement:

$$(\Gamma;\ C_1^{\diamond}\otimes C_2^{\diamond}\otimes\{\rho_{\mathbf{p}}\mapsto S_{\mathbf{p}},\rho_{\mathbf{q}}\mapsto S_{\mathbf{q}},\rho\mapsto U\}) \longrightarrow (\Gamma;\ C_1'^{\diamond}\otimes C_2'^{\diamond}\otimes\{\rho_{\mathbf{p}}\mapsto S_j',\rho_{\mathbf{q}}\mapsto S',\rho\mapsto U\})$$

by induction on the size of $C_1 \otimes C_2$. The base case is given above, while for the inductive case we apply the induction hypothesis use the subtyping relations between the capability sets to conclude by the inductive rule of Definition 4.5.2.

We can now continue proving the main statement, observing:

$(\Gamma;\ C')$ is consistent — by 4.5.1 (4.30)

$\Delta;\Gamma, v_j : \mathtt{tr}(\rho_j), \rho_j : \{\rho_j \mapsto U'_j\} \vdash Q_j; C_1'^{\diamond} \otimes \{\rho_{\mathbf{p}} \mapsto S'_j\}\ (j \in I)$ — premise of [TBR] (4.31)

$\Gamma \vdash v : \mathtt{tr}(\rho); \varnothing$ — premise of [TSEL] (4.32)

$(\Gamma, C_1'^{\diamond} \otimes \{\rho_{\mathbf{p}} \mapsto S'_j\})$ is consistent — the above and 4.5.1 (4.33)

$\Delta;\Gamma \vdash Q_j\{v/v_j\}; C_1'^{\diamond} \otimes \{\rho_{\mathbf{p}} \mapsto S'_j\}$ — the above and 4.5.1 (4.34)

Therefore we conclude by:

$$\frac{\Delta;\Gamma \vdash Q_j\{v/v_j\}; C_1'^{\diamond} \otimes \{\rho_{\mathbf{p}} \mapsto S'_j\} \quad \Delta;\Gamma \vdash Q;\ C_2'^{\diamond} \otimes \{\rho_{\mathbf{q}} \mapsto S'\} \otimes \{\rho \mapsto U'\}}{\Delta;\Gamma \vdash Q_j\{v/v_j\} \mid Q;\ C'}\ [\text{TPAR}]$$

- **case [RCALL]**

We have: $P = \mathbf{def}\, X\langle\tilde{x}\rangle = Q_X\ \mathbf{in}\ (X\langle\tilde{v}\rangle \mid Q) \longrightarrow \mathbf{def}\, X\langle\tilde{x}\rangle = Q_X\ \mathbf{in}\ (Q_X\{\tilde{v}/\tilde{x}\} \mid Q) = P'$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\forall i \in \{1..n\} \qquad \Gamma \vdash v_i : U_i;\ C'_i}{\Delta, X : \tilde{U};\Gamma \vdash X\langle\tilde{v}\rangle;\ C'_X = C'_1 \otimes \ldots \otimes C'_n}\ [\text{TCALL}] \qquad C_X \leqslant C'_X}{\Delta, X : \tilde{U};\Gamma \vdash X\langle\tilde{v}\rangle;\ C_X}\ [\text{TMSUB}] \qquad \Delta, X : \tilde{U};\Gamma \vdash Q;\ C_Q}{\Delta, X : \tilde{U};\Gamma \vdash (X\langle\tilde{v}\rangle \mid Q);\ C}\ [\text{TPAR}] \qquad \vdots}{\Delta;\Gamma \vdash \mathbf{def}\, X\langle\tilde{x}\rangle = Q_X\ \mathbf{in}\ (X\langle\tilde{v}\rangle \mid Q);\ C}\ [\text{TDEF}]$$

Observe that from $\Delta, X : \tilde{U}; \tilde{x} : \tilde{U} \vdash Q_X; \tilde{C}$, by applying Lemma 4.5.1 n times (each time obtaining a consistent context) we obtain $\Delta, X : \tilde{U};\Gamma \vdash Q_X\{\tilde{v}/\tilde{x}\};\ C'_X$. By applying [TMSUB] we obtain $\Delta, X : \tilde{U};\Gamma \vdash Q_X\{\tilde{v}/\tilde{x}\};\ C_X$.

$$\dfrac{\Delta, X : \tilde{U}; \tilde{x} : \tilde{U} \vdash Q_X; \tilde{C} \qquad \dfrac{\Delta, X : \tilde{U};\Gamma \vdash Q_X\{\tilde{v}/\tilde{x}\};\ C_X \quad \Delta, X : \tilde{U};\Gamma \vdash Q;\ C_Q}{\Delta, X : \tilde{U};\Gamma \vdash Q_X\{\tilde{v}/\tilde{x}\} \mid Q;\ C}\ [\text{TPAR}]}{\Delta;\Gamma \vdash \mathbf{def}\, X\langle\tilde{x}\rangle = Q_X\ \mathbf{in}\ (Q_X\{\tilde{v}/\tilde{x}\} \mid Q);\ C}\ [\text{TDEF}]$$

- **case [RRES]** A key case is [RRES], which requires preservation of the condition in [TRES]

that (Γ, C) is consistent. This is because a communication reduction consumes matching prefixes from a pair of dual partial session types, which therefore remain dual. We have $P = (\nu\, s : \Gamma')Q \longrightarrow (\nu\, s)R = P'$, with $Q \longrightarrow R$ (from the rule premise).

$$[\text{TRES}]\ \frac{\Delta;\Gamma,\Gamma' \vdash Q;\ C \otimes C_1 \qquad (\Gamma' = \{\mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho_\mathbf{p}), \rho_\mathbf{p} : \{\rho_\mathbf{p} \mapsto S_\mathbf{p}\}\}_{\mathbf{p}\in I}, C_1 = \otimes_{\mathbf{p}\in I}\{\rho_\mathbf{p} \mapsto S_\mathbf{p}\})}{\Delta;\Gamma \vdash (\nu\, s : \Gamma')Q;\ C}$$

By the induction hypothesis, $\Delta;\Gamma,\Gamma' \vdash R;\ C_2$ and $(\Gamma,\Gamma',C\otimes C_1) \longrightarrow^* (\Gamma,\Gamma',C_2)$.

By Proposition A.1.4 we know that $\mathrm{dom}\,(C_2) = \mathrm{dom}\,(C\otimes C_1) = \mathrm{dom}\,(C \otimes_{\mathbf{p}\in I}\{\rho_\mathbf{p} \mapsto S_\mathbf{p}\})$.

So we can rewrite C_2 as $C'\otimes C_1'$, where $\mathrm{dom}\,(C') = \mathrm{dom}\,(C)$, $\mathrm{dom}\,(C_1') = \mathrm{dom}\,(C_1)$, and $C_1' = \otimes_{\mathbf{p}\in I}\{\rho_\mathbf{p} \mapsto S'_\mathbf{p}\}$.

The reduction $(\Gamma,\Gamma',C\otimes C_1) \longrightarrow^* (\Gamma,\Gamma',C_2)$ then becomes: $(\Gamma,\Gamma',C\otimes C_1) \longrightarrow^* (\Gamma,\Gamma',C'\otimes C_1')$.

By Definition 4.5.2 $(\Gamma,\Gamma',C'\otimes C_1')$ is consistent, and therefore by Corollary 4.5.1 (Γ',C') is consistent. Hence:

$$[\text{TRES}]\ \frac{\Delta;\Gamma,\Gamma' \vdash R;\ C' \otimes C_1' \qquad (\Gamma' = \{\mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho_\mathbf{p}), \rho_\mathbf{p} : \{\rho_\mathbf{p} \mapsto S_\mathbf{p}\}\}_{\mathbf{p}\in I}, C_1' = \otimes_{\mathbf{p}\in I}\{\rho_\mathbf{p} \mapsto S'_\mathbf{p}\})}{\Delta;\Gamma \vdash (\nu\, s : \Gamma')R;\ C'}$$

- **case [RPAR]**

  We have $P = Q \,|\, R \longrightarrow Q' \,|\, R = P'$, with $Q \longrightarrow Q'$ from the rule premise.

$$[\text{TPAR}]\ \frac{\Delta;\Gamma \vdash Q;\ C_1 \qquad \Delta;\Gamma \vdash R;\ C_2}{\Delta;\Gamma \vdash Q \,|\, R;\ C = C_1 \otimes C_2}$$

  By the induction hypothesis $\Delta;\Gamma \vdash Q';\ C_3$, and $(\Gamma, C_1) \longrightarrow^* (\Gamma, C_3)$.

$$[\text{TPAR}]\ \frac{\Delta;\Gamma \vdash Q';\ C_3 \qquad \Delta;\Gamma \vdash R;\ C_2}{\Delta;\Gamma \vdash Q' \,|\, R;\ C_3 \otimes C_2 = C'}$$

  By Corollary 4.5.1 $(\Gamma, C_1 \otimes C_2) \longrightarrow^* (\Gamma, C_3 \otimes C_2)$, which is to say $(\Gamma, C) \longrightarrow^* (\Gamma, C')$ as required.

- **case [RDEF]**

  We have $P = \mathbf{def}\, X\langle \tilde{x} : \tilde{U} \rangle = Q_X \,\mathbf{in}\, Q \longrightarrow \mathbf{def}\, X\langle \tilde{x} : \tilde{U} \rangle = Q_X \,\mathbf{in}\, Q' = P'$ from the rule premise.

$$\frac{\Delta, X : \tilde{U}; \tilde{x} : \tilde{U} \vdash Q_X; \tilde{C} \qquad \Delta, X : \tilde{U}; \Gamma \vdash Q;\ C}{\Delta; \Gamma \vdash \mathbf{def}\, X\langle \tilde{x} : \tilde{U} \rangle = Q_X \,\mathbf{in}\, Q;\ C}\ [\text{TDEF}]$$

  By the induction hypothesis, $\Delta, X : \tilde{U}; \Gamma \vdash Q';\ C'$ and $(\Gamma, C) \longrightarrow^* (\Gamma, C')$, and we conclude by:

$$\frac{\Delta, X : \tilde{U}; \tilde{x} : \tilde{U} \vdash Q_X; \tilde{C} \qquad \Delta, X : \tilde{U}; \Gamma \vdash Q';\ C'}{\Delta; \Gamma \vdash \mathbf{def}\, X\langle \tilde{x} : \tilde{U} \rangle = Q_X \,\mathbf{in}\, Q';\ C'}\ [\text{TDEF}]$$

- **case [RCONG]**

  We have $P \equiv Q$, $P' \equiv Q'$, and $Q \longrightarrow Q'$ from the rule premise. By Proposition 4.5.3 $\Delta; \Gamma \vdash Q;\ C$. By the induction hypothesis $\Delta; \Gamma \vdash Q';\ C'$ and $(\Gamma, C) \longrightarrow^* (\Gamma, C')$. By Proposition 4.5.3 we conclude $\Delta; \Gamma \vdash P';\ C'$.

□

4.5.2 Deadlock Freedom

We show that a typed ensemble of processes interacting on a single session typed by global type G is deadlock-free.

We write the projected typing context of global type G as $\Gamma_G = \{s[\mathbf{p_i}] : S_i\}_{i \in I}$ where for all *iinI* S_i is the projection of G onto $\mathbf{p_i}$.

Theorem 4.5.3 (Deadlock Freedom)**.** *Let* $\varnothing; \varnothing \vdash P; \varnothing$ *with* $P \equiv (\nu\, s : \Gamma_G)|_{i \in I} P_i$ *and each* P_i *interacts only on* $s[\mathbf{p_i}]$*. Then* $P \longrightarrow^* P' \not\longrightarrow$ *implies* $P' \equiv \mathbf{0}$.

Proof. By inversion on [TRES] we obtain $\varnothing; \Gamma_G \vdash P; C_G$. From Theorem 4.5.2 we know that if $P \longrightarrow^* P' \not\longrightarrow$ then there is (Γ'_G, C'_G) such that $(\Gamma_G, C) \longrightarrow (\Gamma'_G, C'_G) \not\longrightarrow$. Then from Definition 3.5.1 if $(\Gamma'_G, C'_G) \not\longrightarrow$ then $(\Gamma'_G(s[\mathbf{p_i}]), C'_G(\rho_i)) = (s[\mathbf{p_i}] : \mathtt{tr}(\rho_i), \rho_i : \{\rho_i \mapsto \mathtt{end}\}; \rho_i : \{\rho_i \mapsto \mathtt{end}\})$. Then from [TINACT] $P' \equiv \mathbf{0}$. □

$$\text{[TCAP]}\ \frac{}{\Gamma \vdash \rho : \{\rho \mapsto S\}; \{\rho \mapsto S\}} \qquad \text{[TVAR]}\ \frac{\mathsf{c} : \mathtt{tr}(\rho), \rho : \{\rho \mapsto S\} \in \Gamma}{\Gamma \vdash \mathsf{c} : \mathtt{tr}(\rho); \varnothing} \qquad \text{[TVAL]}\ \frac{v \in B}{\Gamma \vdash v : B; \varnothing} \qquad \text{[TINACT]}\ \frac{C \text{ terminated}}{\Delta; \Gamma \vdash \mathbf{0}; C}$$

$$\text{[TPAR]}\ \frac{\Delta; \Gamma \vdash P; C_1 \qquad \Delta; \Gamma \vdash Q; C_2}{\Delta; \Gamma \vdash P \mid Q; C_1 \otimes C_2} \qquad \text{[TSUB]}\ \frac{\Delta; \Gamma \vdash P; C \otimes \{\rho \mapsto U\} \qquad U' \leqslant U}{\Delta; \Gamma \vdash P; C \otimes \{\rho \mapsto U'\}}$$

$$\text{[TSEL]}\ \frac{\Gamma \vdash v : U; C \qquad \Delta; \Gamma \vdash P; C' \otimes \{\rho \mapsto S_j\} \qquad \mathsf{c} : \mathtt{tr}(\rho), \rho : \{\rho \mapsto S_j\} \in \Gamma \qquad j \in I}{\Delta; \Gamma \vdash \mathsf{c}[\mathbf{p}] \oplus \langle l_j(v) \rangle . P; C \otimes C' \otimes \{\rho \mapsto \mathbf{p} \oplus_{i \in I} ! l_i(U_i).S_i\}}$$

$$\text{[TBR]}\ \frac{\Delta; \Gamma, x_i : U_i \vdash P_i; C \otimes C_i \otimes \{\rho \mapsto S_i\} \qquad \mathsf{c} : \mathtt{tr}(\rho), \rho : \{\rho \mapsto S_i\} \in \Gamma \qquad \forall i \in I}{\Delta; \Gamma \vdash \mathsf{c}[\mathbf{p}] \& _{i \in I} \{l_i(x_i).P_i\}; C \otimes \{\rho \mapsto \mathbf{p} \ \&_{i \in I} ? l_i(U_i).S_i\}}$$

$$\text{[TSELP]}\ \frac{\Gamma \vdash v : \mathtt{tr}(\rho'); \varnothing \qquad \Delta; \Gamma \vdash P; C \otimes \{\rho \mapsto S_j, \rho' \mapsto U\} \qquad \mathsf{c} : \mathtt{tr}(\rho), \rho : \{\rho \mapsto S_j\} \in \Gamma \qquad j \in I}{\Delta; \Gamma \vdash \mathsf{c}[\mathbf{p}] \oplus \langle l_j(\mathsf{pack}(\rho', v)) \rangle . P; C \otimes \{\rho \mapsto \mathbf{p} \oplus_{i \in I} ! l_i(\exists [\rho' | \{\rho' \mapsto U\}].\mathtt{tr}(\rho')).S_i, \rho' \mapsto U\}}$$

$$\text{[TBRP]}\ \frac{\Delta; \Gamma, \mathtt{v_i} : \mathtt{tr}(\rho_i), \rho_i : \{\rho_i \mapsto U_i\} \vdash P_i; C \otimes \{\rho \mapsto S_i\} \qquad \forall i \in I \qquad \mathsf{c} : \mathtt{tr}(\rho), \rho : \{\rho \mapsto S_i\} \in \Gamma}{\Delta; \Gamma \vdash \mathsf{c}[\mathbf{p}] \& _{i \in I} \{l_i(\mathsf{pack}(\rho_i, \mathtt{v_i})).P_i\}; C \otimes \{\rho \mapsto \mathbf{p} \ \&_{i \in I} ? l_i(\exists [\rho_i | \{\rho_i \mapsto U_i\}].\mathtt{tr}(\rho_i)).S_i\}}$$

$$\text{[TRES]}\ \frac{\Delta; \Gamma, \Gamma' \vdash P; C \otimes C' \qquad (\Gamma' = \{\mathsf{s}[\mathbf{p}] : \mathtt{tr}(\rho_\mathbf{p}), \rho_\mathbf{p} : \{\rho_\mathbf{p} \mapsto S_\mathbf{p}\}\}_{\mathbf{p} \in I}, C' = \otimes_{\mathbf{p} \in I} \{\rho_\mathbf{p} \mapsto S_\mathbf{p}\}) \text{ complete}}{\Delta; \Gamma \vdash (\nu s : \Gamma')P; C}$$

$$\text{[TDEF]}\ \frac{\Delta, X : \tilde{U}; \tilde{x} : \tilde{U} \vdash P; \tilde{C} \qquad \Delta, X : \tilde{U}; \Gamma \vdash Q; C}{\Delta; \Gamma \vdash \mathbf{def}\ X\langle \tilde{x} : \tilde{U} \rangle = P; \tilde{C}\ \mathbf{in}\ Q; C} \qquad \text{[TCALL]}\ \frac{\forall i \in \{1..n\} \qquad \Gamma \vdash v_i : U_i; C_i}{\Delta, X : U_1, \ldots, U_n; \Gamma \vdash X\langle v_1, \ldots, v_n \rangle; C_1 \otimes \ldots \otimes C_n}$$

Figure 4.8: Typing rules

Chapter 5

Case Study

5.1 Producer-Consumer Expanded

The producer-consumer pattern is commonly used in concurrent programming, where multiple threads or processes collaborate to achieve a common goal. It is often used in message queuing systems and event-driven architectures. It is also used in multi-threaded programming to manage the flow of data between threads, to avoid issues such as race conditions and deadlocks. We expand on the producer-consumer scenario from Section 4.1 by discussing the process definitions and showing the typing derivation. To lighten the notation, we present a set of mutually recursive definitions, instead of using the formal syntax of def ... in.

Recall that the example consists of three processes: the producer, the consumer, and a one-place buffer (Figure 4.1). The producer and the consumer communicate with the buffer on a single shared channel. Each of the two must wait to receive the *capability* to communicate on the channel before doing so.

The buffer B is parameterised by channel x and by the capability for it, ρ_x, and alternately responds to add and req messages. At the end of the definition, $\{\rho_x \mapsto S_b\}$ shows the held capability and its session type.

$$\mathsf{B}\langle \mathtt{x} : \mathtt{tr}(\rho_x), \rho_x : \{\rho_x \mapsto S_b\}\rangle = \mathtt{x}[\mathbf{q}]\&\mathsf{add}(\mathsf{i}).\mathtt{x}[\mathbf{q}]\&\mathsf{req}(\mathtt{r}).\mathtt{x}[\mathbf{p}] \oplus \mathsf{snd}(\mathsf{i}).\mathsf{B}\langle \mathtt{x}, \rho_x\rangle; \{\rho_x \mapsto S_b\}$$

The producer is represented by two process definitions: Prd and P. Prd is a recursive process with several parameters. Channels x and y are used to communicate with the consumer and the buffer, respectively. Their capabilities are ρ_x and ρ_y. Finally, i is the value to be sent to the buffer. The producer sends a value to the buffer (add(i)), transfers the capability for the shared channel y (turn(ρ_y)) and receives it back from the consumer. Process P is the entry point for the producer. It has the same parameters as Prd, except for i. The only action of P is to send the consumer a shared reference to the channel used for communication with the buffer

—$\mathsf{x}[\mathbf{c}] \oplus \mathsf{buff}(\mathsf{pack}(\rho_\mathsf{y}, \mathsf{y}[\mathbf{b}]))$.

$$\mathsf{Prd}\langle \mathsf{x} : \mathtt{tr}(\rho_\mathsf{x}), \mathsf{y} : \mathtt{tr}(\rho_\mathsf{y}), \mathsf{i} : \mathtt{Int}, \rho_\mathsf{x} : \{\rho_\mathsf{x} \mapsto S'_\mathbf{p}\}, \rho_\mathsf{y} : \{\rho_\mathsf{y} \mapsto S_\mathbf{q}\}\rangle = \mathsf{y}[\mathbf{b}] \oplus \mathsf{add}(\mathsf{i}).\ \mathsf{x}[\mathbf{c}] \oplus \mathsf{turn}(\rho_\mathsf{y}).\ \mathsf{x}[\mathbf{c}] \& \mathsf{turn}(\rho_\mathsf{y}).\ \mathsf{Prd}\langle \mathsf{x}, \mathsf{y}, \mathsf{i}+1, \rho_\mathsf{x}, \rho_\mathsf{y}\rangle; \{\rho_\mathsf{x} \mapsto S'_\mathbf{p}\} \otimes \{\rho_\mathsf{y} \mapsto S_\mathbf{q}\}$$

$$\mathsf{P}\langle \mathsf{x} : \mathtt{tr}(\rho_\mathsf{x}), \mathsf{y} : \mathtt{tr}(\rho_\mathsf{y}), \rho_\mathsf{x} : \{\rho_\mathsf{x} \mapsto S_\mathbf{p}\}, \rho_\mathsf{y} : \{\rho_\mathsf{y} \mapsto S_\mathbf{q}\}\rangle = \mathsf{x}[\mathbf{c}] \oplus \mathsf{buff}(\mathsf{pack}(\rho_\mathsf{y}, \mathsf{y}[\mathbf{b}]))\ .\ \mathsf{Prd}\langle \mathsf{x}, \mathsf{y}, 0, \rho_\mathsf{x}, \rho_\mathsf{y}\rangle; \{\rho_\mathsf{x} \mapsto S_\mathbf{p}\} \otimes \{\rho_\mathsf{y} \mapsto S_\mathbf{q}\}$$

In a similar way, the consumer is represented by Cons and C. The parameters, however, are different. C has x and its capability ρ_x, for communication with the producer, but it does not have y or ρ_y for communication with the buffer. It receives y from the producer, as part of $\mathsf{pack}(\rho_\mathsf{y}, \mathsf{y}[\mathbf{b}])$, and y is passed as a parameter to Cons. The capability ρ_y is not a parameter of Cons, but it is received in a turn message from the producer.

$$\mathsf{Cons}\langle \mathsf{x} : \mathtt{tr}(\rho_\mathsf{x}), \mathsf{y} : \mathtt{tr}(\rho_\mathsf{y}), \rho_\mathsf{x} : \{\rho_\mathsf{x} \mapsto S'_\mathbf{c}\}\rangle = \mathsf{x}[\mathbf{p}] \& \mathsf{turn}(\rho_\mathsf{y}).\ \mathsf{y}[\mathbf{b}] \oplus \mathsf{req}(r).\ \mathsf{y}[\mathbf{b}] \& \mathsf{snd}(\mathsf{i}).\ \mathsf{x}[\mathbf{p}] \oplus \mathsf{turn}(\rho_\mathsf{y}).\ \mathsf{Cons}\langle \mathsf{x}, \mathsf{y}, \rho_\mathsf{x}\rangle; \{\rho_\mathsf{x} \mapsto S'_\mathbf{c}\}$$

$$\mathsf{C}\langle \mathsf{x} : \mathtt{tr}(\rho_\mathsf{x}), \rho_\mathsf{x} : \{\rho_\mathsf{x} \mapsto S_\mathbf{c}\}\rangle = \mathsf{x}[\mathbf{p}] \& \mathsf{buff}(\mathsf{pack}(\rho_\mathsf{y}, \mathsf{y}[\mathbf{b}]))\ .\ \mathsf{Cons}\langle \mathsf{x}, \mathsf{y}, \rho_\mathsf{x}\rangle; \{\rho_\mathsf{x} \mapsto S_\mathbf{c}\}$$

The complete system consists of the producer, the consumer, and the buffer in parallel, with sessions s_1 (roles $\mathbf{p}$ and $\mathbf{c}$) and s_2 (roles $\mathbf{q}$ and $\mathbf{b}$) scoped to construct a closed process.

$$(\nu\, s_1)((\nu\, s_2)(\mathsf{P}\langle \mathsf{s}_1[\mathbf{p}], \mathsf{s}_2[\mathbf{q}], \rho_\mathbf{p}, \rho_\mathbf{q}\rangle \mid \mathsf{B}\langle \mathsf{s}_2[\mathbf{b}], \rho_\mathbf{b}\rangle) \mid \mathsf{C}\langle \mathsf{s}_1[\mathbf{c}], \rho_\mathbf{c}\rangle$$

The session types involved in these processes are projections of the global type G (Section 4.3). They specify how each role is expected to use its channel endpoint. The roles are $\mathbf{b}$ for the buffer, $\mathbf{q}$ for the combined role of the producer and the consumer as they interact with the buffer, $\mathbf{p}$ for the producer, and $\mathbf{c}$ for the consumer.

$$S_\mathbf{b} = G \upharpoonright \mathbf{b} = \mu \mathtt{t}.\mathbf{q}?\mathsf{add}(\mathtt{Int}).\mathbf{q}?\mathsf{req}(\mathtt{Str}).\mathbf{q}!\mathsf{snd}(\mathtt{Int}).t$$

$$S_\mathbf{q} = G \upharpoonright \mathbf{q} = \mu \mathtt{t}.\mathbf{b}!\mathsf{add}(\mathtt{Int}).\mathbf{b}!\mathsf{req}(\mathtt{Str}).\mathbf{b}?\mathsf{snd}(\mathtt{Int}).t$$

$$S_\mathbf{p} = G \upharpoonright \mathbf{p} = \mathbf{c}!\mathsf{buff}(\exists[\rho_\mathbf{q} | \{\rho_\mathbf{q} \mapsto S'_\mathbf{q}\}].\mathtt{tr}(\rho_\mathbf{q})).\mu \mathtt{t}.\mathbf{c}!\mathsf{turn}(\{\rho_\mathbf{q} \mapsto S'_\mathbf{q}\}).\mathbf{c}?\mathsf{turn}(\{\rho_\mathbf{q} \mapsto S_\mathbf{q}\}).t$$

$$S_\mathbf{c} = G \upharpoonright \mathbf{c} = \mathbf{p}?\mathsf{buff}(\exists[\rho_\mathbf{q} | \{\rho_\mathbf{q} \mapsto S'_\mathbf{q}\}].\mathtt{tr}(\rho_\mathbf{q})).\mu \mathtt{t}.\mathbf{p}?\mathsf{turn}(\{\rho_\mathbf{q} \mapsto S'_\mathbf{q}\}).\mathbf{p}!\mathsf{turn}(\{\rho_\mathbf{q} \mapsto S_\mathbf{q}\}).t$$

These types occur in the capabilities associated with each process. For example process $\mathsf{P}\langle \mathsf{s}_1[\mathbf{p}], \mathsf{s}_2[\mathbf{q}], \rho_\mathbf{p}, \rho_\mathbf{q}\rangle$ has $\{\rho_\mathbf{q} \mapsto S_\mathbf{q}\} \otimes \{\rho_\mathbf{p} \mapsto S_\mathbf{p}\}$, process $\mathsf{B}\langle \mathsf{s}_2[\mathbf{b}], \rho_\mathbf{b}\rangle$ has $\{\rho_\mathbf{b} \mapsto S_\mathbf{b}\}$, and process $\mathsf{C}\langle \mathsf{s}_1[\mathbf{c}], \rho_\mathbf{c}\rangle$ has $\{\rho_\mathbf{c} \mapsto S_\mathbf{c}\}$.

To illustrate the typing rules, we show the typing derivation for the producer, i.e. processes

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash \rho_{\mathbf{q}} : \{\rho_{\mathbf{q}} \mapsto S'_{\mathbf{q}}\}; \{\rho_{\mathbf{q}} \mapsto S'_{\mathbf{q}}\}}\,[\text{TCAP}] \quad \mathrm{x} : \mathtt{tr}(\rho_{\mathrm{x}}), \rho_{\mathrm{x}} : \{\rho_{\mathrm{x}} \mapsto \mathbf{c}?\mathsf{turn}(\{\rho_{\mathbf{q}} \mapsto S_{\mathbf{q}}\}).S'_{\mathbf{p}}\} \in \Gamma}{\Delta; \Gamma \vdash \mathrm{x}[\mathbf{c}] \oplus \mathsf{turn}(\rho_{\mathbf{q}}).\mathrm{x}[\mathbf{c}]\&\mathsf{turn}(\rho_{\mathrm{y}}).\mathsf{Prd}\langle x, y, \mathrm{i}+1, \rho_{\mathrm{x}}, \rho_{\mathrm{y}}\rangle; \{\rho_{\mathrm{x}} \mapsto S'_{\mathbf{p}}, \rho_{\mathrm{y}} \mapsto S'_{\mathbf{q}}\}}\,[\text{TSEL}] \quad \dfrac{\mathrm{i} \in \mathtt{Int}}{\Gamma \vdash \mathrm{i} : \mathtt{Int}; \varnothing}\,[\text{TVAL}] \quad \mathrm{y} : \mathtt{tr}(\rho_{\mathrm{y}}), \rho_{\mathrm{y}} : \{\rho_{\mathrm{y}} \mapsto S'_{\mathbf{q}}\} \in \Gamma}{\Delta; \Gamma \vdash \mathrm{y}[\mathbf{b}] \oplus \mathsf{add}(\mathrm{i}).\mathrm{x}[\mathbf{c}] \oplus \mathsf{turn}(\rho_{\mathbf{q}}).\mathrm{x}[\mathbf{c}]\&\mathsf{turn}(\rho_{\mathbf{q}}).\mathsf{Prd}\langle \mathrm{x}, \mathrm{y}, \mathrm{i}+1, \rho_{\mathrm{x}}, \rho_{\mathrm{y}}\rangle; \{\rho_{\mathrm{x}} \mapsto S'_{\mathbf{p}}, \rho_{\mathrm{y}} \mapsto S_{\mathbf{q}}\}}\,[\text{TSEL}]$$

Typing Derivation for Prd

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash \mathrm{x} : \mathtt{tr}(\rho_{\mathrm{x}}), \mathrm{y} : \mathtt{tr}(\rho_{\mathrm{y}}), \rho_{\mathrm{x}} : \{\rho_{\mathrm{x}} \mapsto S'_{\mathbf{p}}\}, \rho_{\mathrm{y}} : \{\rho_{\mathrm{y}} \mapsto S_{\mathbf{q}}\}; \{\rho_{\mathrm{x}} \mapsto S'_{\mathbf{p}}, \rho_{\mathrm{y}} \mapsto S_{\mathbf{q}}\}}\,[\text{TCALL}]}{\Delta; \Gamma \vdash \mathsf{Prd}\langle x, y, \mathrm{i}, \rho_{\mathbf{p}}, \rho_{\mathbf{q}}\rangle; \{\rho_{\mathrm{x}} \mapsto S'_{\mathbf{p}}, \rho_{\mathrm{y}} \mapsto S_{\mathbf{q}}\}} \quad \dfrac{\mathrm{y} : \mathtt{tr}(\rho_{\mathrm{y}}), \rho_{\mathrm{y}} : \{\rho_{\mathrm{y}} \mapsto S_{\mathbf{q}}\}}{\Gamma \vdash \mathrm{y} : \mathtt{tr}(\rho_{\mathrm{y}}); \varnothing}\,[\text{TVAR}] \quad \mathrm{x}, \rho_{\mathrm{x}} : \{\rho_{\mathrm{x}} \mapsto S_{\mathbf{p}}\} \in \Gamma}{\Delta; \Gamma \vdash \mathrm{x}[\mathbf{c}] \oplus \mathsf{buff}(\mathsf{pack}(\rho_{\mathbf{q}}, \mathrm{y}[\mathbf{b}])).\mathsf{Prd}\langle \mathrm{x}, \mathrm{y}, \mathrm{i}, \rho_{\mathrm{x}}, \rho_{\mathrm{y}}\rangle; \{\rho_{\mathrm{y}} \mapsto \mathbf{p}!\mathsf{buff}(\exists[\rho_{\mathrm{y}} | \{\rho_{\mathrm{y}} \mapsto S_{\mathbf{q}}\}].\mathtt{tr}(\rho_{\mathrm{y}})).S'_{\mathbf{p}}, \rho_{\mathrm{y}} \mapsto S_{\mathbf{q}}\}}\,[\text{TSELP}]$$

Typing Derivation for P

Figure 5.1: Typing Derivation for Producer

Prd and P in Figure 5.1. The derivations use the following type and context definitions.

$$\begin{aligned} S'_{\mathbf{p}} &= \mu t.\mathbf{c}!\mathsf{turn}(\{\rho_{\mathbf{q}} \mapsto S'_{\mathbf{q}}\}).\mathbf{c}?\mathsf{turn}(\{\rho_{\mathbf{q}} \mapsto S_{\mathbf{q}}\}).t \\ S'_{\mathbf{q}} &= \mathbf{b}!\mathsf{req}(\mathtt{Str}).\mathbf{b}?\mathsf{snd}(\mathtt{Int}).S_{\mathbf{q}} \\ \Delta &= \mathsf{Prd} : (\mathtt{tr}(\rho_{\mathbf{p}}), \mathtt{tr}(\rho_{\mathbf{q}}), \mathtt{Int}, \{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}\}, \{\rho_{\mathbf{q}} \mapsto S_{\mathbf{q}}\}) \\ \Gamma &= \mathrm{x} : \mathtt{tr}(\rho_{\mathbf{p}}), \mathrm{y} : \mathtt{tr}(\rho_{\mathbf{q}}), \mathrm{i} : \mathtt{Int}, \rho_{\mathbf{p}} : \{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}\}, \rho_{\mathbf{q}} : \{\rho_{\mathbf{q}} \mapsto S_{\mathbf{q}}\} \end{aligned}$$

We show the typing derivation for the producer, i.e. processes Cons and C in Figure 5.2. The derivations use the following type and context definitions.

$$\begin{aligned} \Delta &= \mathsf{C} : (\mathtt{tr}(\rho_{\mathrm{x}}), \{\rho_{\mathrm{x}} \mapsto S\}), \mathsf{Cons} : (\mathtt{tr}(\rho_{\mathrm{x}}), \mathtt{tr}(\rho_{\mathrm{y}}), \{\rho_{\mathrm{x}} \mapsto S\}) \\ \Gamma' &= \Gamma, \mathrm{y} : \mathtt{tr}(\rho_{\mathrm{y}}), \rho_{\mathrm{y}} : \{\rho_{\mathrm{y}} \mapsto S'_{\mathbf{q}}\} \\ S'_{\mathbf{c}} &= \mu t.\mathbf{p}?\mathsf{turn}(\{\rho_{\mathrm{y}} \mapsto S'_{\mathbf{q}}\}).\mathbf{p}!\mathsf{turn}(\{\rho_{\mathrm{y}} \mapsto S_{\mathbf{q}}\}).t \end{aligned}$$

Finally we show the typing derivation for the buffer, B (Figure 5.3).

5.2 One Producer Two Consumers

Scenarios with multiple producers/consumers can be represented in a similar way, the capabilities acting as a form of lock for the resource being shared. We extend our example with a second

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma', \mathtt{i} : \mathtt{Int} \vdash \rho_y; \{\rho_y \mapsto S_q\}}\,[\text{TCAP}] \quad \mathtt{x} : \mathtt{tr}(\rho_x), \rho_x : \{\rho_x \mapsto S_c\} \in \Gamma' \quad \vdots}{\Delta; \Gamma', \mathtt{i} : \mathtt{Int} \vdash \mathtt{x}[\mathbf{p}] \oplus \mathsf{turn}(\rho_y).\mathsf{Cons}\langle x, y, \rho_x\rangle; \{\rho_x \mapsto \mathbf{p}!\mathsf{turn}(\{\rho_y \mapsto S_q\}).S_c, \rho_y \mapsto S_q\}}\,[\text{TSEL}]}{\Delta; \Gamma' \vdash \mathtt{y}[\mathbf{q}] \& \mathsf{snd}(\mathtt{i}).\mathtt{x}[\mathbf{p}] \oplus \mathsf{turn}(\rho_y).\mathsf{Cons}\langle x, y, \rho_x\rangle; \{\rho_x \mapsto \mathbf{p}!\mathsf{turn}(\{\rho_y \mapsto S_q\}).S_c, \rho_y \mapsto \mathbf{b}?\mathsf{snd}(\mathtt{Int}).S_q\}}\,[\text{TBR}] \quad \vdots}{\Delta; \Gamma' \vdash \mathtt{y}[\mathbf{b}] \oplus \mathsf{req}(r).\mathtt{y}[\mathbf{b}] \& \mathsf{snd}(\mathtt{i}).\mathtt{x}[\mathbf{p}] \oplus \mathsf{turn}(\rho_y).\mathsf{Cons}\langle x, y, \rho_x\rangle; \{\rho_x \mapsto \mathbf{p}!\mathsf{turn}(\{\rho_y \mapsto S_q\}).S_c\} \otimes \{\rho_y \mapsto \mathbf{b}!\mathsf{req}(\mathtt{Str}).\mathbf{b}?\mathsf{snd}(\mathtt{Int}).S_q\}}\,[\text{TSEL}]}{\Delta, \Gamma \vdash \mathtt{x}[\mathbf{p}] \& \mathsf{turn}(\rho_y).\mathtt{y}[\mathbf{b}] \oplus \mathsf{req}(r).\mathtt{y}[\mathbf{b}] \& \mathsf{snd}(\mathtt{i}).\mathtt{x}[\mathbf{p}] \oplus \mathsf{turn}(\rho_y).\mathsf{Cons}\langle x, y, \rho_x\rangle; \{\rho_x \mapsto S'_c\}}\,[\text{TBR}]$$

[TVAL] $\mathtt{r} \in \mathtt{Str}$ / $\Gamma' \vdash r : \mathtt{Str}; \varnothing$; $\mathtt{y}, \rho_y : \{\rho_y \mapsto S_q\} \in \Gamma'$

Typing Derivation for Cons

$$\dfrac{\dfrac{\dfrac{}{\Delta; \Gamma, \mathtt{y} : \mathtt{tr}(\rho_y), \rho_y : \{\rho_y \mapsto S'_q\} \vdash x : \mathtt{tr}(\rho_x), y : \mathtt{tr}(\rho_y), \rho_x : \{\rho_x \mapsto S'_c\}; \{\rho_x \mapsto S'_c\}}\,[\text{TCALL}]}{\Delta; \Gamma, \mathtt{y} : \mathtt{tr}(\rho_y), \rho_y : \{\rho_y \mapsto S'_q\} \vdash \mathsf{Cons}\langle x, y, \rho_x\rangle; \{\rho_x \mapsto S'_c\}} \quad \vdots \quad \mathtt{x} : \mathtt{tr}(\rho_x), \rho_x : \{\rho_x \mapsto S'_c\} \in \Gamma}{\Delta; \Gamma \vdash \mathtt{x}[\mathbf{p}] \oplus \langle \mathsf{buff}(\mathsf{pack}(\rho_y, \mathtt{y}))\rangle.\mathsf{Cons}\langle x, y, \rho_x\rangle; \{\rho_x \mapsto \mathbf{c}?l(\exists[\rho_y | \{\rho_y \mapsto S'_q\}].\mathtt{tr}(\rho_y)).S'_c\}}\,[\text{TBRP}]$$

Typing Derivation for C

Figure 5.2: Typing Derivation for Consumer

consumer. The producer and the two consumers communicate with the buffer via a single shared channel. Each of the three must wait to receive the *capability* to communicate on the channel before doing so.

$$\begin{aligned} G = \mathbf{p} \to \mathbf{c}_1 &: \mathsf{buff}(\exists[\rho_q | \{\rho_q \mapsto S'_q\}].\mathtt{tr}(\rho_q)).\mathbf{p} \to \mathbf{c}_2 : \mathsf{buff}(\exists[\rho_q | \{\rho_q \mapsto S'_q\}].\mathtt{tr}(\rho_q)). \\ &\mu \mathtt{t}.\mathbf{q} \to \mathbf{b} : \mathsf{add}(\mathtt{Int}).\mathbf{p} \to \mathbf{c}_1 : \mathsf{turn}(\{\rho_q \mapsto S'_q\}).\mathbf{q} \to \mathbf{b} : \mathsf{req}(\mathtt{Str}).\mathbf{b} \to \mathbf{q} : \mathsf{snd}(\mathtt{Int}). \\ &\mathbf{c}_1 \to \mathbf{p} : \mathsf{turn}(\{\rho_q \mapsto S_q\}).\mathbf{q} \to \mathbf{b} : \mathsf{add}(\mathtt{Int}).\mathbf{p} \to \mathbf{c}_2 : \mathsf{turn}(\{\rho_q \mapsto S'_q\}).\mathbf{q} \to \mathbf{b} : \mathsf{req}(\mathtt{Str}). \\ &\mathbf{b} \to \mathbf{q} : \mathsf{snd}(\mathtt{Int}).\mathbf{c}_2 \to \mathbf{p} : \mathsf{turn}(\{\rho_q \mapsto S_q\}).\mathtt{t} \end{aligned}$$

It captures the interaction between the producer and consumer entities through roles $\mathbf{p}$, $\mathbf{c}_1$, $\mathbf{c}_2$ and between producer, the two consumers and buffer through roles $\mathbf{q}$ (shared between producer and the two consumers) and $\mathbf{b}$. Projecting onto each of the roles gives the session types:

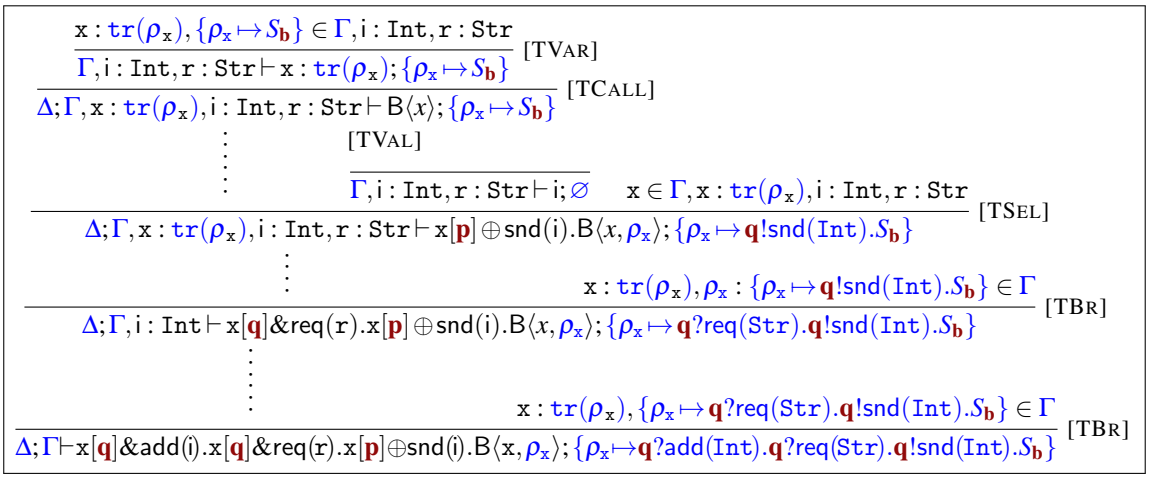

Figure 5.3: Typing Derivation for B

$$S_\mathbf{b} = G \upharpoonright \mathbf{b} = \mu t.\mathbf{q}?\mathsf{add}(\mathtt{Int}).\mathbf{q}?\mathsf{req}(\mathtt{Str}).\mathbf{q}!\mathsf{snd}(\mathtt{Int}).\mathbf{q}?\mathsf{add}(\mathtt{Int}).\mathbf{q}?\mathsf{req}(\mathtt{Str}).\mathbf{q}!\mathsf{snd}(\mathtt{Int}).t$$

$$S_\mathbf{q} = G \upharpoonright \mathbf{q} = \mu t.\mathbf{b}!\mathsf{add}(\mathtt{Int}).\mathbf{b}!\mathsf{req}(\mathtt{Str}).\mathbf{b}?\mathsf{snd}(\mathtt{Int}).\mathbf{b}!\mathsf{add}(\mathtt{Int}).\mathbf{b}!\mathsf{req}(\mathtt{Str}).\mathbf{b}?\mathsf{snd}(\mathtt{Int}).t$$

$$S_\mathbf{p} = G \upharpoonright \mathbf{p} = \mathbf{c_1}!\mathsf{buff}(\exists[\rho_\mathbf{q} | \{\rho_\mathbf{q} \mapsto S'_\mathbf{q}\}].\mathtt{tr}(\rho_\mathbf{q})).\mathbf{c_2}!\mathsf{buff}(\exists[\rho_\mathbf{q} | \{\rho_\mathbf{q} \mapsto S'_\mathbf{q}\}].\mathtt{tr}(\rho_\mathbf{q})).$$
$$\mu t.\mathbf{c_1}!\mathsf{turn}(\{\rho_\mathbf{q} \mapsto S'_\mathbf{q}\}).\mathbf{c_1}?\mathsf{turn}(\{\rho_\mathbf{q} \mapsto S_\mathbf{q}\}).\mathbf{c_2}!\mathsf{turn}(\{\rho_\mathbf{q} \mapsto S'_\mathbf{q}\}).\mathbf{c_2}?\mathsf{turn}(\{\rho_\mathbf{q} \mapsto S_\mathbf{q}\}).t$$

$$S_{\mathbf{c_1}} = G \upharpoonright \mathbf{c_1} = \mathbf{p}?\mathsf{buff}(\exists[\rho_\mathbf{q} | \{\rho_\mathbf{q} \mapsto S'_\mathbf{q}\}].\mathtt{tr}(\rho_\mathbf{q})).\mu t.\mathbf{p}?\mathsf{turn}(\{\rho_\mathbf{q} \mapsto S'_\mathbf{q}\}).\mathbf{p}!\mathsf{turn}(\{\rho_\mathbf{q} \mapsto S_\mathbf{q}\}).t$$

$$S_{\mathbf{c_2}} = G \upharpoonright \mathbf{c_2} = \mathbf{p}?\mathsf{buff}(\exists[\rho_\mathbf{q} | \{\rho_\mathbf{q} \mapsto S'_\mathbf{q}\}].\mathtt{tr}(\rho_\mathbf{q})).\mu t.\mathbf{p}?\mathsf{turn}(\{\rho_\mathbf{q} \mapsto S'_\mathbf{q}\}).\mathbf{p}!\mathsf{turn}(\{\rho_\mathbf{q} \mapsto S_\mathbf{q}\}).t$$

The buffer process then becomes:

$$\mathsf{B}\langle \mathtt{x} : \mathtt{tr}(\rho_\mathtt{x}), \rho_\mathtt{x} : \{\rho_\mathtt{x} \mapsto S_\mathbf{b}\} \rangle = \mathtt{x}[\mathbf{q}] \& \mathsf{add}(\mathtt{i}).\mathtt{x}[\mathbf{q}] \& \mathsf{req}(\mathtt{r}).\mathtt{x}[\mathbf{p}] \oplus \mathsf{snd}(\mathtt{i}).$$
$$\mathtt{x}[\mathbf{q}] \& \mathsf{add}(\mathtt{i}).\mathtt{x}[\mathbf{q}] \& \mathsf{req}(\mathtt{r}).\mathtt{x}[\mathbf{p}] \oplus \mathsf{snd}(\mathtt{i}).\mathsf{B}\langle \mathtt{x}, \rho_\mathtt{x} \rangle; \{\rho_\mathtt{x} \mapsto S_\mathbf{b}\}$$

As before, the buffer B is parameterised by channel x and by the capability for it, $\rho_\mathtt{x}$, and alternately responds to add and req messages. At the end of the definition, $\{\rho_\mathtt{x} \mapsto S_\mathbf{b}\}$ shows the held capability and its session type.

As previously, the producer is represented by two process definitions: Prd and P. The difference being that it now communicates with a second consumer.

$$\mathsf{Prd}\langle \mathtt{x} : \mathtt{tr}(\rho_\mathtt{x}), \mathtt{y} : \mathtt{tr}(\rho_\mathtt{y}), \mathtt{i} : \mathtt{Int}, \rho_\mathtt{x} : \{\rho_\mathtt{x} \mapsto S'_\mathbf{p}\}, \rho_\mathtt{y} : \{\rho_\mathtt{y} \mapsto S_\mathbf{q}\} \rangle = \mathtt{y}[\mathbf{b}] \oplus \mathsf{add}(\mathtt{i}).$$
$$\mathtt{x}[\mathbf{c_1}] \oplus \mathsf{turn}(\rho_\mathtt{y}).\ \mathtt{x}[\mathbf{c_1}] \& \mathsf{turn}(\rho_\mathtt{y}).\ \mathtt{y}[\mathbf{b}] \oplus \mathsf{add}(\mathtt{i}).\ \mathtt{x}[\mathbf{c_2}] \oplus \mathsf{turn}(\rho_\mathtt{y}).\ \mathtt{x}[\mathbf{c_2}] \& \mathsf{turn}(\rho_\mathtt{y}).$$
$$\mathsf{Prd}\langle \mathtt{x}, \mathtt{y}, \mathtt{i}{+}1, \rho_\mathtt{x}, \rho_\mathtt{y} \rangle; \{\rho_\mathtt{x} \mapsto S'_\mathbf{p}\} \otimes \{\rho_\mathtt{y} \mapsto S_\mathbf{q}\}$$

$$\mathsf{P}\langle \mathtt{x} : \mathtt{tr}(\rho_\mathtt{x}), \mathtt{y} : \mathtt{tr}(\rho_\mathtt{y}), \rho_\mathtt{x} : \{\rho_\mathtt{x} \mapsto S_\mathbf{p}\}, \rho_\mathtt{y} : \{\rho_\mathtt{y} \mapsto S_\mathbf{q}\} \rangle = \mathtt{x}[\mathbf{c_1}] \oplus \mathsf{buff}(\mathsf{pack}(\rho_\mathtt{y}, \mathtt{y}[\mathbf{b}])).$$
$$\mathtt{x}[\mathbf{c_2}] \oplus \mathsf{buff}(\mathsf{pack}(\rho_\mathtt{y}, \mathtt{y}[\mathbf{b}])).\ \mathsf{Prd}\langle \mathtt{x}, \mathtt{y}, 0, \rho_\mathtt{x}, \rho_\mathtt{y} \rangle; \{\rho_\mathtt{x} \mapsto S_\mathbf{p}\} \otimes \{\rho_\mathtt{y} \mapsto S_\mathbf{q}\}$$

Each consumer is represented as before. We show only the definition for one of them, C_1.

$$\mathsf{Cons}_1\langle \mathtt{x} : \mathtt{tr}(\rho_{\mathtt{x}}), \mathtt{y} : \mathtt{tr}(\rho_{\mathtt{y}}), \rho_{\mathtt{x}} : \{\rho_{\mathtt{x}} \mapsto S'_{\mathbf{c}}\}\rangle = \mathtt{x}[\mathbf{p}]\&\mathsf{turn}(\rho_{\mathtt{y}}).\ \mathtt{y}[\mathbf{b}] \oplus \mathsf{req}(r).\ \mathtt{y}[\mathbf{b}]\&\mathsf{snd}(\mathsf{i}).\ \mathtt{x}[\mathbf{p}] \oplus \mathsf{turn}(\rho_{\mathtt{y}}).\ \mathsf{Cons}_1\langle \mathtt{x}, \mathtt{y}, \rho_{\mathtt{x}}\rangle; \{\rho_{\mathtt{x}} \mapsto S'_{\mathbf{c}_1}\}$$

$$\mathsf{C}_1\langle \mathtt{x} : \mathtt{tr}(\rho_{\mathtt{x}}), \rho_{\mathtt{x}} : \{\rho_{\mathtt{x}} \mapsto S_{\mathbf{c}_1}\}\rangle = \mathtt{x}[\mathbf{p}]\&\mathsf{buff}(\mathsf{pack}(\rho_{\mathtt{y}}, \mathtt{y}[\mathbf{b}])).\ \mathsf{Cons}_1\langle \mathtt{x}, \mathtt{y}, \rho_{\mathtt{x}}\rangle; \{\rho_{\mathtt{x}} \mapsto S_{\mathbf{c}_1}\}$$

The complete system consists of the producer, the two consumers, and the buffer in parallel, with sessions s_1 (roles $\mathbf{p}$, $\mathbf{c_1}$, and $\mathbf{c_2}$ and s_2 (roles $\mathbf{q}$ and $\mathbf{b}$) scoped to construct a closed process.

$$(\nu\, s_1)((\nu\, s_2)(\mathsf{P}\langle \mathtt{s}_1[\mathbf{p}], \mathtt{s}_2[\mathbf{q}], \rho_{\mathbf{p}}, \rho_{\mathbf{q}}\rangle \mid \mathsf{B}\langle \mathtt{s}_2[\mathbf{b}], \rho_{\mathbf{b}}\rangle) \mid \mathsf{C}\langle \mathtt{s}_1[\mathbf{c_1}], \rho_{\mathbf{c_1}}\rangle \mid \mathsf{C}\langle \mathtt{s}_1[\mathbf{c_2}], \rho_{\mathbf{c_2}}\rangle$$

Chapter 6

Discussion

Since session types were first introduced, channel endpoints are usually treated as linear resources, meaning that they can be used only once and must be consumed after use. This approach ensures that each role in a protocol can be implemented by a unique agent, as each agent will hold a distinct channel endpoint that corresponds to its role. This treatment of channel endpoints as linear resources is reinforced by several connections between session types and other linear type theories: the encodings of binary session types and multiparty session types into linear π-calculus types [39, 122]; the Curry-Howard correspondence between binary session types and linear logic [20, 139]; the connection between multiparty session types and linear logic [23, 24].

Some session type systems go beyond the traditional linear typing discipline and allow for more flexible usage of session channels. Vasconcelos [133] allows a session type to become non-linear, and shareable, when it reaches a state that is invariant with every subsequent message. Mostrous and Vasconcelos [95] define *affine* session types, in which each endpoint must be used at most once and can be discarded with an explicit operator. In Fowler *et al.*'s [56] implementation of session types for the Links web programming language, affine typing allows sessions to be cancelled when exceptions (including dropped connections) occur. Caires and Pérez [19] use monadic types to describe cancellation (i.e. affine sessions) and non-determinism. Pruiksma and Pfenning [120] use adjoint logic to describe session cancellation and other behaviours including multicast and replication.

When a data structure contains a linear value, the linearity of that value spreads to the entire data structure. Consumption of the linear value requires the entire data structure to be consumed as well. For example, consider a data structure that contains a linear channel endpoint. Since the channel endpoint is linear, it can be used only once and must be consumed after use. Therefore, the entire data structure must also be consumed after the channel endpoint is used. This means that the data structure itself must also be linear, since it is consumed along with the linear channel endpoint. In the standard π-calculus exceptions to this spreading nature of linearity have been studied by Kobayashi in his work on deadlock-freedom and by Padovani [110], who extends the linear π-calculus with composite regular types in such a way that data containing linear values

can be shared among several processes. However, this sharing can occur only if there is no overlapping access to such values, which differs from our work where we have full sharing of values. On the other hand, we work directly with (multiparty) session types, whereas Padovani works with linear π-calculus and obtains his results via the encoding of session types into linear π-types [39].

Session types are related to the concept of *typestate* [127], especially in the work of Kouzapas *et al.* [85, 86] which defines a typestate system for Java based on multiparty session types. Typestate systems require linear typing or some other form of alias control, to avoid conflicting state changes via multiple references. Approaches include the permission-based systems used in the Plural and Plaid languages [13, 128] and the fine-grained approach of Militão *et al.* [93]. Crafa and Padovani [36] develop a "chemical" approach to concurrent typestate oriented programming, allowing objects to be accessed and modified concurrently by several processes, each potentially changing only part of their state. Our approach is partly inspired by Fähndrich and DeLine's "adoption and focus" system [53], in which a shared stateful resource (in our case, a session channel) is separated from the linear key (capability, in our system) that enables it to be used. In this way the state changes of channels follow the standard session operations, channels can be shared (for example, stored in shared data structures), and access can be controlled by passing the capability around the system.

The work by Balzer *et al.* [6, 7] is the closest to this one. It supports sharing of binary session channels by allowing locks to be acquired and released at points that are explicitly specified in the session type. Our approach with multiparty sessions is not based on locks, so it doesn't require runtime mechanisms for managing blocked processes and notifying them when locks are released.

We have presented a new system of multiparty session types with capabilities, which allows sharing of resources in a way that generalises the strictly linear or affine access control typical of session type systems. The key technical idea is to separate a channel from the capability of using the channel. This allows channels to be shared, while capabilities are linearly controlled. We use a form of existential typing to maintain the link between a channel and its capability, while both are transmitted in messages. This allows the system to ensure that only parties that have the appropriate capability can use a given channel, while still allowing for sharing of channels among multiple parties. We have proved communication safety, formulated as a subject reduction theorem (Theorem 4.5.2).

Part III

Typechecking Java Protocols with [St]Mungo

Chapter 7

[St]Mungo toolchain: An overview

7.1 Introduction

In this chapter we introduce the [St]Mungo, a toolchain based on the relation between session types and typestates.

Typestates [127] define sequences of operations that are permitted for an entity in a particular state. The type of an entity is associated with a partially ordered set of typestates. When the entity's state changes, its typestate may change as well. Operations on entities are correct if the resulting typestate is reachable by a typestate transition from the previous typestate (following the order). Typestates can be used to restrict valid parameters, return values, field values and thus provide guarantees on an entity's behaviour. Typestates can be modelled using finite-state machines, so in statically typed languages we can check at compile time whether all possible sequences of operations are valid with respect to the correct use of the entity.

Since their introduction [127], there have been several projects to add typestate to practical languages. Vault [41, 53] is an extension of C, and Fugue [42] applies similar ideas to C#. Plural [14] is based on Java and has been used to study access control systems [12] and transactional memory [8], and to evaluate the effectiveness of typestate in Java APIs [14]. Sing# [51] is an extension of C# which was used to implement Singularity, an operating system based on message-passing. It incorporates typestate-like contracts, which are a form of session type, to specify protocols. Bono *et al.* [21] have formalised a core calculus based on Sing# and proved type safety. Aldrich *et al.* [1, 128] propose a new paradigm of typestate-oriented programming, implemented in the Plaid language. Instead of class definitions, a program consists of state definitions containing methods that cause transitions to other states. Transitions are specified in a similar way to Plural's pre- and post-conditions. Like classes, states are organised into an inheritance hierarchy. Recent work [58, 143] uses gradual typing to integrate static and dynamic typestate checking. Bodden and Hendren [16] developed the Clara framework, which combines static typestate analysis with runtime monitoring. The monitoring is based on the trace matches approach [2], using regular expressions to define allowed sequences of method calls. The static

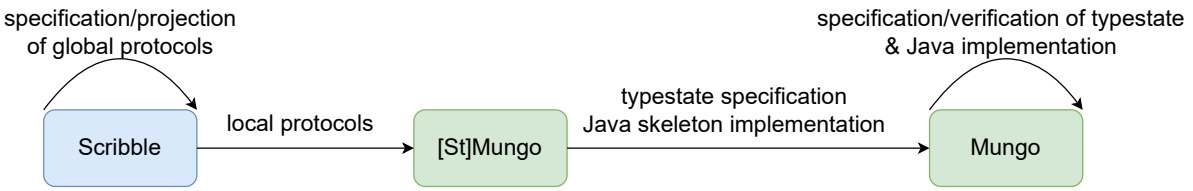

Figure 7.1: [St]Mungo toolchain workflow

analysis attempts to remove the need for runtime monitoring, but if this is not possible, the runtime monitor is optimised.

In [64] Gay *et al.*use the concept of typestates to integrate binary session types and object-oriented programming. They define a translation from the session type of a communication channel endpoint into a typestate specification that constrains the use of send and receive methods on an object representing the channel endpoint.

Kouzapas *et al.* [85, 86] have extended this work to define a typestate system for Java based on multiparty session types, implemented as the [St]Mungo toolchain. Mungo is a front-end typechecking tool, that extends Java with typestate definitions, by allowing classes to be associated with a definition of the permitted sequences of method calls. Typestate specifications however cannot represent the notion of duality of session types, so the notion of consistency from multiparty session type theory does not translate. Compatibility between roles depends on the assumption that their typestate specifications are derived from a single global session type. The second tool introduced by Kouzapas *et al.*, StMungo ("Scribble-to-Mungo") translates from Scribble local protocols into typestate specifications and Java program skeletons. These local protocols are projected from a global protocol after being validated by the Scribble tools. This two tool approach ensures compatibility between roles, as well as allowing the full implementation to be checked statically for any errors.

7.2 StMungo

The StMungo tool [38, 85, 86, 137] is a Java-based transpiler implemented using the ANTLR v4.5 framework [3]. StMungo acts as a bridge between multiparty session types and typestate specifications. In particular it is the link between the Scribble specification language [70, 126] and the Mungo tool Section 7.3. StMungo is the first tool to provide a practical embedding of Scribble multiparty session types into an object-oriented language with typestates.

In order to better understand the StMungo tool, we need to describe both the Scribble language and the typestate specifications. Let's start with Scribble.

The Scribble specification language is an implementation of multiparty session types (MPST) [72, 126]. Participants in a distributed system communicate among each other by sending and receiving messages and following a predefined communication protocol. Such a protocol is given

as a *global protocol* (or global type) in Scribble. The Scribble tools can perform *validation* and *projection* of a global protocol. First, we must check if the specified global protocol is valid, meaning if it is correct with respect to transmitted data; there are no deadlocks within the global protocol; there are no un-notified participants for example, regarding session termination, and so on. These checks follow the MPST theory [72]. Once a global protocol is validated, with Scribble tools we can project it into *local protocols* (or local types) for each participant in the system.

Let us illustrate the notions of global and local protocols using the two-buyer example, from Chapter 3 Figure 3.1.

```
1  module examples.twobuyer.TwoBuyer;
2  type <java> "java.lang.String" from "java.lang.String" as Str;
3  type <java> "java.lang.Integer" from "java.lang.Integer" as Int;
4  global protocol TwoBuyer(role Buyer1, role Buyer2, role Seller) {
5      title(Str) from Buyer1 to Seller;
6      quote(Int) from Seller to Buyer1;
7      quote(Int) from Seller to Buyer2;
8      rec Negotiate{
9          split(Int) from Buyer1 to Buyer2;
10         choice at Buyer2 {
11             ok() from Buyer2 to Buyer1;
12             ok() from Buyer2 to Seller;
13             transfer(Int) from Buyer1 to Seller;
14             transfer(Int) from Buyer2 to Seller;
15             shipping(Str) from Seller to Buyer1;
16             shipping(Str) from Seller to Buyer2; }
17         or {
18             no() from Buyer2 to Buyer1;
19             stillnegotiating() from Buyer2 to Seller;
20             continue Negotiate; }
21         or {
22             quit() from Buyer2 to Buyer1;
23             quit() from Buyer2 to Seller;}}}
```

Listing 7.1: TwoBuyer Global Protocol in Scribble

The global protocol for the two-buyer protocol specified in Scribble is given in Listing 7.1. Line 1 contains the `module` declaration, made up of an optional package prefix i.e., `examples.twobuyer`, and the name of the file containing the module, `TwoBuyer`. Line 3 contains a payload type declaration `type <java>...`, which gives an alias (`Str`) to a data type (`String`) from an external language `java` which can be used in the payload of a message signature. A module can contain zero or more *global protocol declarations*, consisting of a protocol signature (line 4), message passing (line 5), choices (line 10), and recursion (line 8). A message signature, `title(Str) from Buyer1 to Seller`, consists of an operator name which acts as a message identifier i.e. `title` and zero or more payload types, in this case a single `Str`.

A choice (e.g., `choice at Buyer2`) states the subject role, `Buyer2`, for which selecting one of the cases, separated by `or`, is a mutually exclusive internal choice. A do statament enacts the

specified protocol, and can be used for recursive definitions.

Using the Scribble tools, we can project the TwoBuyer global protocol onto local protocols for the buyers `Buyer1` and `Buyer2`, and the seller `Seller`.

The local protocol for `Buyer1`, given in Listing 7.2, describes the behaviour of this role. The `_Buyer1` in the protocol name indicates that `Buyer1` is the local endpoint.

```
1  ...
2  local protocol TwoBuyer_Buyer1(self Buyer1, role Buyer2, role Seller){
3      title(Str) to Seller;
4      quote(Int) from Seller;
5      rec Negotiate {
6          split(Int) to Buyer2;
7          choice at Buyer2 {
8              ok() from Buyer2;
9              transfer(Int) to Seller;
10             shipping(Str) from Seller; }
11         or {
12             no() from Buyer2;
13             continue Negotiate; }
14         or { quit() from Buyer2;}}}
```

Listing 7.2: TwoBuyer Buyer1 Protocol – obtained by using Scribble to project the global protocol onto role Buyer1

The StMungo tool takes as input a Scribble local protocol for a `role` and translates it into a typestate specification for a Java API skeleton. This translation is based on the principle that each `role` in the multiparty session communication following its local protocol, can be abstracted as a Java class following its typestate specification. A typestate is a state machine defining the permitted sequence of method calls of a Java object, thus defining the object's protocol.

Running StMungo on the Buyer1 protocol Listing 7.2 produces the following files:

1. `Buyer1Protocol.protocol`: the typestate specification representing the Buyer1's local protocol. The send and receive operations are translated as Java methods (Listing 7.3 below in this section).

2. `Buyer1Role.java`: the Java API implementing Buyer1. This class implements the typestate `Buyer1Protocol` over Java sockets (Listing 7.4, Section 7.3).

3. `Buyer1Main.java`: this can be an optional file. It gives a minimum logic of the client `CRole` and provides a `main()` method (Listing 7.5 in Section 7.3).

The typestate specification `Buyer1Protocol.protocol` for Buyer1 is given in Listing 7.3.

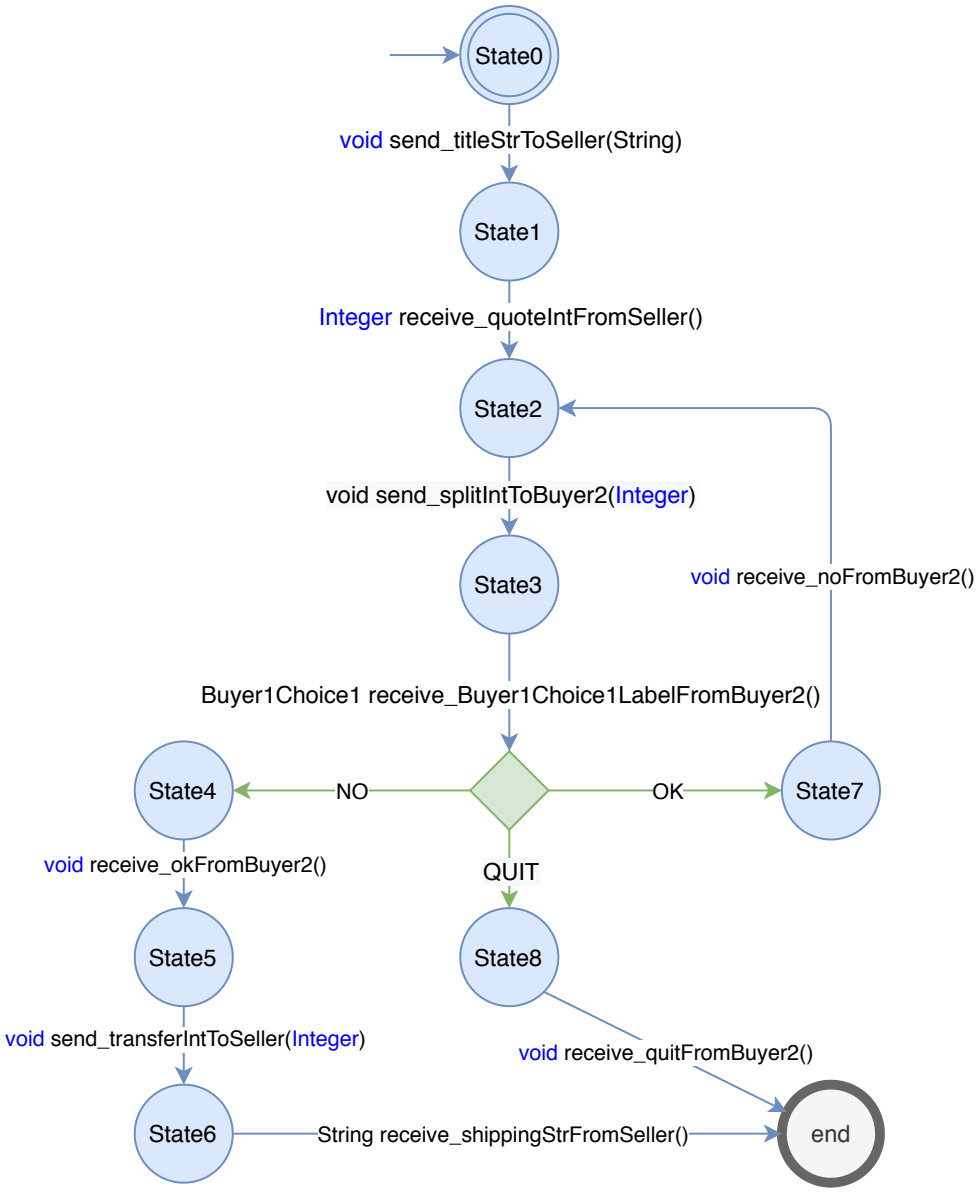

Figure 7.2: State Machine for Buyer1Protocol

```
typestate Buyer1Protocol {
    State0 = {void send_titleStrToSeller(String): State1}
    State1 = {Integer receive_quoteIntFromSeller(): State2}
    State2 = {void send_splitIntToBuyer2(Integer): State3}
    State3 = {Buyer1Choice1 receive_Buyer1Choice1LabelFromBuyer2():
                <OK: State4, NO: State7, QUIT: State8>}
    State4 = {void receive_okFromBuyer2(): State5}
    State5 = {void send_transferIntToSeller(Integer): State6}
    State6 = {String receive_shippingStrFromSeller(): end}
    State7 = {void receive_noFromBuyer2(): State2}
    State8 = {void receive_quitFromBuyer2(): end}}
```

Listing 7.3: Buyer1 typestate

A typestate is a state machine (Figure7.2) with states labelled `State0` (initial state), `State1`,

`State2` ... Each state offers a set of methods that must be a subset of the methods defined by the class; each method specifies a transition to a successor state, such that when called at runtime allows the object to change state as specified by its typestate.

The send and receive operations given in the local protocols for Buyer1 are translated as typestate methods in `Buyer1Protocol.protocol`. For example, the message `title(Str) to Seller` (line 4, Listing 7.2) where Buyer1 sends a `title` message of type `Str` to the Seller, is translated as a method call (line 2 in Listing 8.3). For each choice there is an enumerated type, named according to the numerical position of the choice in the sequence of choices within the local protocol. The values of the enumerated type are the names of the first message in each branch of the choice. For the choice in `TwoBuyerLoop_Buyer1` we have the following definition.

```
1 enum Buyer1Choice1 { OK, NO, QUIT; }
```

For each role involved in the choice there wll be an enumerated type with the same set of values, but the names of the types are not the same for every role.

We will comment on the other two files `Buyer1Role.java` and `Buyer1Main.java` in Section 7.3.

To improve the translation of the Scribble language to typestates some extensions have been implemented by the author:

- translation of messages with no payload, i.e. `message_operator()`
- translation of messages with multiple payload, i.e.
  `message_operator(payload_type`$_1$`, ..., payload_type`$_n$`)`
- translation of messages without a message signature, i.e.
  `(payload_type`$_1$`, ..., payload_type`$_n$`)`
- translation of messages with annotated payloads, i.e.
  `message_operator(annotation : payload_type)`
- translation of special cases of recursions nested in choice structures. A simple example of a problematic Scribble specification is:

```
1 global protocol ProtocolName(role S, role C) {
2   choice at C { rcpt(String) from C to S; }
3           or { msg(String) from C to S;
4                rec loop { subject(String) from C to S;
5                           continue loop; }}}
```

- translation of special cases of nested choice inside a recursion. A simple example of a problematic Scribble specification is:

```
1 global protocol ProtocolName(role S, role C) {
2   command(String) from C to S;
```

```
3   rec loop{ choice at S {
4                           ok(String) from S to C;
5                           choice at S { end(String) from S to C;}
6                                    or { sum(String) from S to C;}
7                           message(String) from S to C; }
8                   or {    error(String) from S to C; }
9           continue loop; }}
```

- translation of the `do` statement, a construct that instructs the specified roles to perform the interactions of the specified protocol inline with the current protocol,
  i.e. `do ProtocolName(role`$_1$`,..., role`$_n$`)`

7.3 Mungo

The Mungo tool [38, 85, 86, 137] is a Java front-end tool used to statically typecheck typestate specifications for Java classes. The tool is implemented in Java using the ExtendJ framework [67, 105], a meta-compiler based on reference attribute grammars.

Mungo extends a Java class with a typestate specification, which is saved in a separate file (such as `Buyer1Protocol.protocol` in Section 7.2) and is attached to a Java class using the annotation `@Typestate("ProtocolName")`, where `"ProtocolName"` names the file where the typestate is defined. The typestate inference algorithm given by the formalisation of the Mungo tool in [85, 86] constructs the sequences of methods called on all objects associated with a typestate, and then checks if the inferred typestate is a subtype of the object's declared typestate.

Source files are typechecked in two phases: first, according to the standard Java type system, and then to the typestate type system via Mungo. The source files can then be compiled using standard `javac` and executed in the standard Java runtime environment.

The typestate specification generated from StMungo together with the Mungo typechecker can guide the user in the design and development of distributed multiparty communication-based systems with guarantees of communication safety and soundness.

We will now describe the use of Mungo via the `TwoBuyer` running example, and in particular we will do so by commenting on the last two files `Buyer1Role.java` and `Buyer1Main.java` generated by StMungo for the `Buyer1` role.

The outline for Buyer1 is given by Listing 7.4 annotated by the typestate `Buyer1Protocol`, defined in Listing 7.3. Lines 3–9 define `Buyer1`'s constructor where the connection phase over Java sockets takes place. The rest of `Buyer1Role` contains a minimal implementation of the methods specified in the typestate `Buyer1Protocol`. The methods for sending and receiving messages contain basic formatting and parsing, which can be further improved by the programmer. When instantiated it establishes socket connections to the other roles in the session (`Buyer2Role` and `SellerRole`).

```
@Typestate("Buyer1Protocol")
public class Buyer1Role{
  public Buyer1Role() {...//Bind the sockets and accept a client
    connection
      try { // Create the read and write streams
          socketSellerIn = new BufferedReader(..);
          socketSellerOut = new PrintWriter(..);
      }catch (IOException e) {
          System.out.println("Read failed"); System.exit(-1);}}
  public void send_titleStrToSeller(String payload) {
      this.socketSellerOut.println(payload);}
  public Integer receive_quoteIntFromSeller() {
      String line = "";
      try { line = this.socketSellerIn.readLine();}
      catch(IOException e) {
          System.out.println("Input/Output error. "); System.exit(-1);}
      return Integer.parseInt(line);}
  public void send_splitIntToBuyer2(Integer payload) {
      this.socketBuyer2Out.println(payload0);}
  public Buyer1Choice1 receive_Buyer1Choice1LabelFromBuyer2() {
      String stringLabelBuyer1Choice1 = "";
      try { stringLabelBuyer1Choice1 = this.socketBuyer2In.readLine();}
      catch(IOException e) {
          System.out.println("Input/Output error, unable to get label.");
          System.exit(-1);}
      switch(stringLabelBuyer1Choice1) {
          case "OK":
              return Buyer1Choice1.OK;
          case "NO":
              return Buyer1Choice1.NO;
          case "QUIT":
          default:
              return Buyer1Choice1.QUIT;}}
  ... // Define all other methods in Buyer1Protocol}
```

Listing 7.4: Buyer1 Role

Let's move now onto the `Buyer1Main.java` given in Listing 7.5. `Buyer1Main.java` contains a minimal implementation of the client endpoint using the `Buyer1Role` class to communicate with the server endpoint. Below we give the main method, omitting any auxiliary methods generated by StMungo.

```
1  public static void main(String[] args) {
2    Buyer1Role currentBuyer1 =  new Buyer1Role();
3    BufferedReader readerBuyer1 = new BufferedReader(..);
4    currentBuyer1.send_titleStrToSeller(safeRead(readerBuyer1));
5    System.out.println("Received"+currentBuyer1.receive_quoteIntFromSeller
       ());
6
7    _Negotiate: do{
8      Integer payload5 = Integer.parseInt(safeRead(readerBuyer1));
9      currentBuyer1.send_splitIntToBuyer2(payload5);
10     switch(currentBuyer1.receive_Buyer1Choice1LabelFromBuyer2()) {
11       case OK:
12         currentBuyer1.receive_okFromBuyer2();
13         currentBuyer1.send_transferIntToSeller(Integer.parseInt(safeRead(
       readerBuyer1)));
14         String payload10 = currentBuyer1.receive_shippingStrFromSeller();
15         break _Negotiate;
16       case NO:
17         currentBuyer1.receive_noFromBuyer2();
18         continue _Negotiate;
19       case QUIT:
20         currentBuyer1.receive_quitFromBuyer2();
21         break _Negotiate;}} while(true);}
```

Listing 7.5: Buyer1 Main

To ensure that methods of the protocol are called in a valid sequence and that all possible responses are handled, the `Buyer1Main` implementation is checked by computing the sequences of method calls that are made on the `currentBuyer1` object, inferring the minimal typestate specification that allows them, and then comparing it with the specification declared in `Buyer1Protocol`.

The following work has been undertaken on the Mungo tool by the author:

- the tool has been moved from the Java 1.4 compiler to the Java 1.8 compiler and adapted to work with the new framework. The codebase of the tool has been updated to use Java 1.8 language features and APIs. This required changes to the syntax and structure of the code, as well as rewriting parts of it to take advantage of new features. Finally it has been tested to ensure that it works correctly with the new compiler.

- synchronised statements, the conditional operator ?:, inner and anonymous classes, and static initialisers. Mungo was extended to allow synchronised statements and the conditional operator ?: to be used on an object with a typestate specification as long as the behaviour of the object is consistent with the specification. Mungo's typechecking was extended to analyse the correctness and safety of transitions caused by operations of inner classes, as well as checking the interactions between the inner and the outer class.

- the tool has been extended to support the full use of language for classes without typestate definition.

- a special typestate annotation has been implemented, through which typestate specifications are associated with Java classes, without any change to the language itself. This required changes to Mungo and the research compiler used by Mungo to to add an annotation which allows typestate specifications to be associated with Java classes. The file name specified in the annotation is resolved relative to the location of the class file. The typechecker ensures that instances of that class will follow the specified protocol. This allows the programs to be used both with the Mungo tool and with a regular Java compiler, where the annotation and typestate specification will be ignored.

- inheritance between protocols. Inheritance between protocols (or subtyping) allows a protocol to inherit the behaviour of another protocol and extend or modify it as needed. This is useful for creating a hierarchy of protocols, where a more general protocol is defined at a higher level and more specific protocols are defined at lower levels. The subtype of the typestate may add new operations that are only valid in a certain state, or remove operations that are not valid in it. Subtypes may also override methods from their supertype to implement different behaviour for the same operation. Mungo was extended to account for inheritance between protocols using the typical Java keyword `extends` followed by the name of the supertype, and allow object of the subtype typestate to be used in place of objects of its supertype. Finally, the typechecker was extended to analyse the resulting typestate state machine to ensure that the sequence of operations is correct and safe.

Chapter 8

Real-World Case Studies

8.1 Introduction

In this chapter we present three real world case studies to show the applicability of the [St]Mungo toolchain to typecheck protocols. In the following sections we highlight the benefits and some of the difficulties of representing these.

The first two case studies presented, HTTP and FTP, represent common internet protocols. A long standing aim of session types is to represent internet protocols, and instances of these protocols often appear as examples in the literature from the very first papers [71]. Internet protocols are prone to certain errors that can be excluded by specifying and constraining the communication behaviour using session type systems.

These errors tends to fall into the following categories:

- *communication mismatches*: when the message sent is not one expected by the receiver.
- *deadlock*: when some set of participants are all blocked on mutually dependent input actions.
- *orphan messages*: when the receiver terminates without reading an incoming message.

The final case-study presented in this chapter is basic Paxos, a protocol for solving consensus in a network of unreliable agents. Consensus is the process of agreeing on one result among a group of participants. The different setting of this protocol pushes the boundaries of what session types can represent and offers insight into what is needed to enhance the specifying and verification power of session types.

With each of the case studies we demonstrate some of the strengths of the toolchain, and disscuss improvements that would lead to a more fine grained representation of the protocols, and in some cases better interoperability with existing implementations.

The full implementation of these examples can be found in the mungo tools repository at https://bitbucket.org/abcd-glasgow/mungo-tools/src/master/.

8.2 HTTP

HTTP (HyperText Transfer Protocol) [54] is the underlying data protocol used by the World Wide Web defining how messages are formatted and transmitted, and what actions servers and clients may take in response to various methods, such as `GET`, `PUT` or `POST`. An HTTP session is a sequence of network request-response transactions, initiated by the client sending a request over a TCP connection to a particular port of a server. Upon receiving the request, the server listening on that port sends back a message containing a status line, such as "HTTP/1.1 200 OK", and aditional information. The structure of the request and response messages exchanged is rich and complex, involving branching, recursion or optional headers, lending itself to be further specified through session types. Hence, we represent the HTTP global protocol in the style of Hu [74] where some of the data structure is moved onto protocol structure. A monolithic HTTP *request* is broken down into several smaller messages: sending request line – `GET / HTTP/1.1`, followed by sending zero or more header-fields – `Host: www.google.co.uk` or `Accept: text/html, application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8` and terminated by a sending new-line. A *response* is similarly broken down into: a status line `HTTP/1.1 200 OK`, followed by zero or more header-fields – `Date: Sun, 24 May 2019 10:04:36 GMT` or `Server: Apache` and terminated by a new-line. This fine grained representation of the protocol is made possible by the messages being broken down via TCP bit streams, in a manner that is transparent to both parties involved.

In this section, we apply the [St]Mungo toolset to the `GET` method of HTTP/1.1 [54]. We represent a minimal number of message fields required for interoperability with existing real-world servers.

A snippet of the global protocol specifing HTTP in Scribble is given in Listing 8.1. The client's *request* to the server is represented between lines 4 and 12, starting with a message requesting the resouce, `request(str) from C to S`. Within the recursion `rec X` the client can choose as many header fields as needed, ending by sending the `body message`. The server's *response* is specified between line 14 and line 24, starting with the status line made up of `httpv` and a `choice` of status, followed by zero or more header-fields such as `date` or `server` and terminated by a new-line.

```
...
global protocol Http(role C, role S){
  request(str) from C to S;  //GET / HTTP/1.1
  rec X{ choice at C { host(str) from C to S;//Host: www.google.co.uk
                       continue X;}
                  or { userA(str) from C to S;//User-Agent:...
                       continue X;}
                  or { acceptT(str) from C to S;//Accept: text/html...
                       continue X;}
                  or { ... //other header fields
                      body(str) from C to S;}}
  //Response
  httpv(str) from S to C;//HTTP/1.1
  choice at S { 200(str) from S to C; } //200 OK
           or { 404(str) from S to C; } //404 Bad Request
  rec Y{ choice at S { date(str) from S to C;//Date: ...
                       continue Y;}
                  or { server(str) from S to C;//Server:...
                       continue Y;}
                  or { strictTS(str) from S to C;//Strict-Transport-Security
                       continue Y;}
                  or { ...//other header fields
                       body(str) from S to C;}}}
```

Listing 8.1: HTTP Global Protocol in Scribble

Using the Scribble tools, we can project the HTTP global protocol onto local protocols for the server `S` and the client `C`.

The local protocol for the HTTP client `C`, an extract of which is given in Listing 8.2, describes the behaviour of this role. The `_C` in the protocol name indicates that `C` is the local endpoint.

```
...
local protocol Http_C(role C, role S) {
  request(str) to S;
  rec X { choice at C { host(str) to S; continue X; }
                  or { userA(str) to S; continue X; }
                  or { acceptT(str) to S; continue X; }
                  or { ...//other header fields
                       body(str) to S;}}
  httpv(str) from S;
  choice at S { 200(str) from S; }
           or { 404(str) from S; }
  rec Y { choice at S { date(str) from S; continue Y; }
                  or { server(str) from S; continue Y; }
                  or { strictTS(str) from S; continue Y; }
                  or { ...//other header fields
                        body(str) from S; }}}
```

Listing 8.2: HTTP Client Protocol – obtained by using Scribble to project the global protocol onto role C

The client will send a request line `request` (line), followed by zero or more header-fields — `host`, or `userA` and so on. It will expect to receive a response with a status line containing the

HTTP version — `httpv` (line 18) followed by the status of the request, either — `200` for a found resource, or — `404` for a bad request; and any additional information the server may send.

Running StMungo on the HTTP client protocol Listing 8.2 produces the following files, where `C` at the beginning of each file name stands for client.

1. `CProtocol.protocol`: the typestate specification representing the HTTP client's local protocol. The send and receive operations are translated as Java methods (Listing 8.3 below in this section).

2. `CRole.java`: the Java API implementing the HTTP client. This class implements the typestate `CProtocol` over Java sockets (Listing 8.4, Section7.3).

3. `CMain.java`: this can be an optional file. It gives a minimum logic of the client `CRole` and provides a `main()` method (Listing 8.5, Section7.3).

An excerpt of the typestate specification `CProtocol.protocol` for the HTTP client is given in Listing 8.3.

```
1  typestate CProtocol {
2    State0 = { void send_requestStrToS(String): State1 }
3    State1 = { void send_HOSTToS(): State2,
4               void send_USERAToS(): State3,
5               void send_ACCEPTTToS(): State4,
6               ... //send other labels
7               void send_BODYToS(): State11 }
8    State2 = { void send_hostStrToS(String): State1 }
9    State3 = { void send_userAStrToS(String): State1 }
10   State4 = { void send_acceptTStrToS(String): State1 }
11   ... //send other main messages
12   State11 = { void send_bodyStrToS(String): State12 }
13   State12 = { String receive_httpvStrFromS(): State13 }
14   State13 = { CChoice1 receive_CChoice1LabelFromS():
15                 <_200: State14, _404: State15> }
16   State14 = { String receive_200StrFromS(): State16 }
17   State15 = { String receive_404StrFromS(): State16 }
18   State16 = { CChoice2 receive_CChoice2LabelFromS(): <DATE: State17,
19                 SERVER: State18, STRICTTS: State19, ..., BODY: State34> }
20   State17 = { String receive_dateStrFromS(): State16 }
21   State18 = { String receive_serverStrFromS(): State16 }
22   State19 = { String receive_strictTSStrFromS(): State16 }
23   ...
24   State34 = { String receive_bodyStrFromS(): end } }
```

Listing 8.3: HTTP Client Typestate Specification

The send and receive operations given in the client's local protocol are translated as typestate methods in `CProtocol.protocol`. For example, the message `request(str) to S` (line 4, Listing 8.2) where the client sends a `request` message of type `str` to the server, is translated as `void send_requestStrToS(String)` (line 2 in Listing 8.3). A visual representation of the

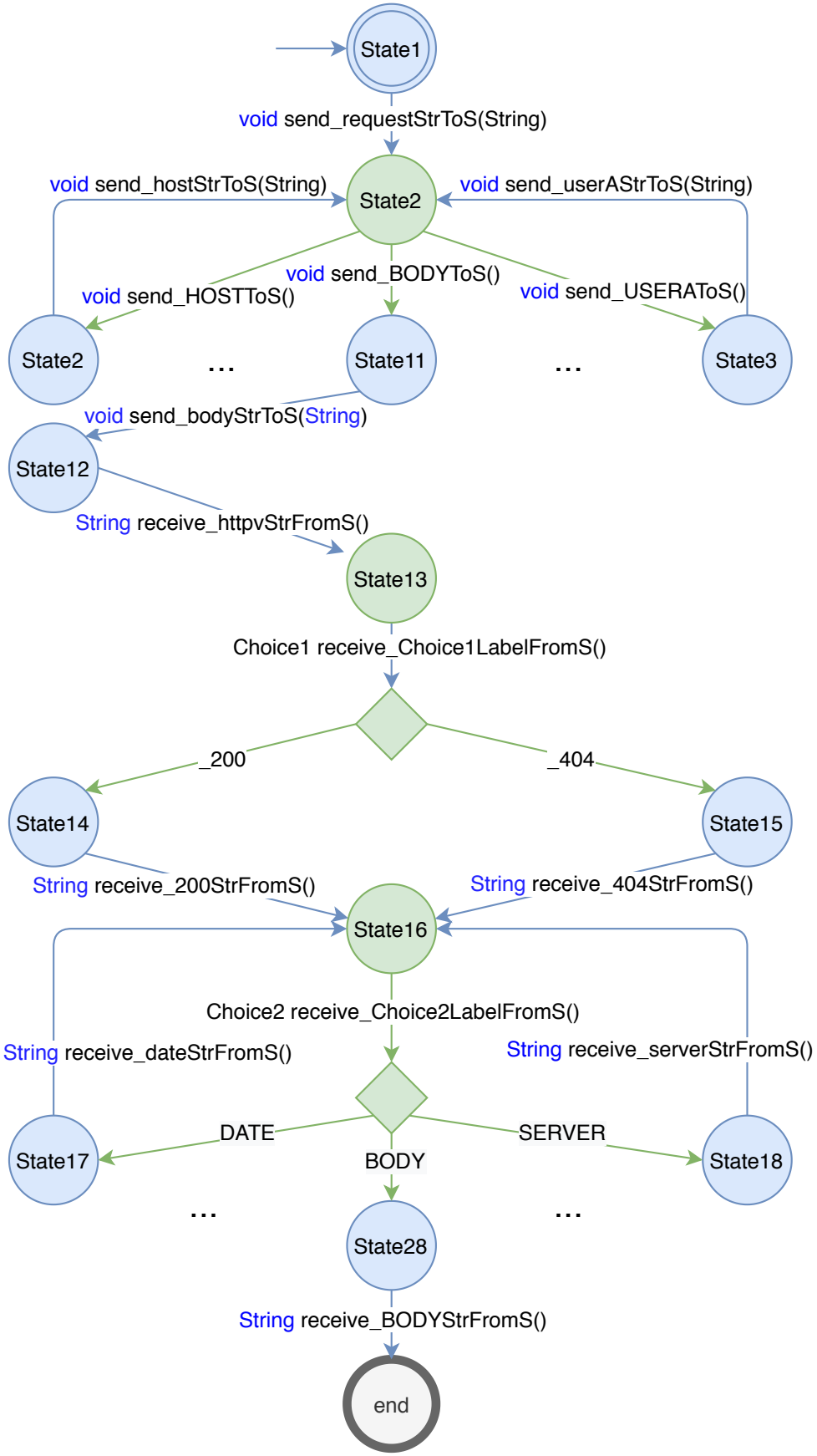

Figure 8.1: State Machine for CProtocol

typestate as a state machine is given in Figure 8.1.

```
@Typestate("CProtocol")
public class CRole {
  public CRole() { ...//Bind the sockets and accept a client connection
      try { // Create the read and write streams
          socketSIn = new BufferedReader(...);
          socketSOut = new PrintWriter(...);}
      catch (IOException e) {...}
    }
  public void send_requestStrToS(String payload){
      this.socketSOut.println(payload);
    }
  ... // Define all other send methods in CProtocol
  public String receive_httpvStrFromS()() {
      String line = "";
      try {line = this.socketSIn.readLine();}
      catch (IOException e) {...}
      return line;
    }
  public Choice1 receive_Choice1LabelFromS() {
      try {stringLabelChoice1 = this.socketSIn.readLine();}
      catch (IOException e) {...}
      switch (stringLabelChoice1) {
        case "200":
          return Choice1._200;
        case "404":
        default:
          return Choice1._404;
      }}
  public String receive_200StrFromS() {
      String line = "";
      try {line = this.socketSIn.readLine();}
      catch (IOException e) {...}
      return line;
    }
  public String receive_404StrFromS() {
      String line = "";
      try {line = this.socketSIn.readLine();}
      catch (IOException e) {...}
      return line;
    } .../*Define all other receive methods in CProtocol*/}}}

```

Listing 8.4: HTTP Client API

The HTTP client API is given by Listing 8.4 annotated by the typestate `CProtocol`, defined

in Listing 8.3. Lines 3–9 define the client's constructor where the connection phase over Java sockets takes place. The rest of `CRole` contains a minimal implementation of the methods specified in the typestate `CProtocol`. Line 9 defines the method for sending the initial, mandatory, request line `send_requestStrToS`. Lines 13–34 define methods for receiving the first line in a response, composed of the HTTP version — `receive_httpvStrFrom` and the status. The method in line 19 `Choice1 receive_Choice1LabelFromS` captures the status. This method returns a `Choice1` type, which is an enumerated type defined as:

```
1 enum Choice1 {_200, _404;}
```

The values of the enumerated type are the names of the first message in each branch of the choice, i.e. `_200` or `_404`. Thus, the method `receive_Choice1LabelFromS` receives a message which represents one of the two status codes, and it returns the corresponding `enum` value. All methods for sending and receiving messages contain basic formatting and parsing, which can be further improved by the programmer.

Let's move now onto the `CMain.java` given in Listing 8.5. `CMain.java` contains a minimal implementation of the client endpoint using the CRole class to communicate with the server endpoint. Below we give the main method, omitting the auxiliary methods generated by StMungo. The code is modified from the generated version by adding the request and host messages needed to request the home page from `www.google.co.uk`.

```
1  public static void main(String[] args) {
2    CRole currentC = new CRole();
3    currentC.send_requestStrToS("GET / HTTP/1.1");
4    _X: do {
5      String sread = //input header choice
6      switch (sread) {
7        case ("HOST"):
8          currentC.send_HOSTToS();
9          currentC.send_hostStrToS("www.google.co.uk");
10         continue _X;
11        ... //other cases corresponding to header fields
12       case ("BODY"):
13         currentC.send_BODYToS();
14         currentC.send_bodyStrToS("/r/n");
15         break _X;
16       }} while (true);
17   currentC.receive_httpvStrFromS();
18   switch (currentC.receive_Choice1LabelFromS()) {
19     case _200:
20       currentC.receive_200StrFromS();
21       break;
22     case _404:
23       currentC.receive_404StrFromS();
24       break;}
25 _Y:do {
26   switch (currentC.receive_Choice2LabelFromS()) {
27     case DATE:
28       currentC.receive_dateStrFromS();
29       continue _Y;
30     ... //other cases corresponding to the header fields
31     case BODY:
32       currentC.receive_bodyStrFromS();
33       break _Y;}
34   } while (true);}
```

Listing 8.5: HTTP Client Implementation

In line 2 we create a new HTTP client, `currentC`, and proceed by showing the code for a small correctly formatted request, with the initial, mandatory request line messages being sent first (line 3); then among the recursive choice cases we show the code for sending the the host field (lines 7–10), before concluding the request by an empty body (lines 12-15). Then `currentC` will receive the response status line (lines 17–24) followed by recursive choice cases for the fields to be received from the server (lines 25–33).

To ensure that methods of the protocol are called in a valid sequence and that all possible responses are handled, the `CMain` implementation is checked by computing the sequences of method calls that are made on the `currentC` object, inferring the minimal typestate specification

that allows them, and then comparing it with the specification declared in `CProtocol`. This ensures that the client implementation follows the fine grained protocol structure specified.

The typestate and skeleton implementation allow the programmer to take advantage of the safety properties offered by session types, and ensure the correctness of the request/response messages while preserving protocol interoperability. The typestate specification can also act as a form of documentation for the protocol, summarising several dozen RFC pages. This implementation can be combined with one of the many Java libraries for HTTP, such as Apache HttpComponents[1], to make use of the functionality already provided by them.

[1] http://hc.apache.org

8.3 FTP

The File Transfer Protocol (FTP) [119] is a standard internet protocol used to transfer files between a client and a server. FTP uses a basic command-reply mechanism. The client connects to the server, and begins a synchronized conversation by sending a command, to which the server will respond, signaling readiness for the next command. Server responses come in a standardized format: a 3-digit response code followed by a message. The codes have the same general meanings, though the exact message that follows may vary. The first digit of the response code is the most important, as it is an indicator of the overall success or failure status of the command. Generally response codes follow these rules: 1, or 2 if the command was successful, 3 if an additional action is required; or 4 or 5 if the command failed.

For example, if the client were to issue the USER command the server could reply with a 230 "User logged in, proceed" response, indicating that the authentication has been successful. Alternatively, the server could reply with a 331 "User name okay, need password" prompting the client for further input. The server might also reply with a 500-level command, such as 530 "Not logged in" to indicate that authentication has been unsuccessful.

Once the client has successfully established a connection with the server and passed authentication, it can then attempt to retrieve or upload a file. In a typical session where files are transferred, FTP will use two separate connections: the control and data connections.

The **control connection**, typically established on port 21, is the primary connection and is used for sending commands back and forth between the client and server. After establishing a connection, the client can issue the retrieve command, RETR, to initiate the file transfer, followed by the name of the file to be retrieved. If the file exists and if the client has rights to access the file, the server will issue a reply indicating that everything is OK and that the file transfer will now begin.

Using the established control connection, the client and server will create a separate **data connection**, used only for transferring the requested data. Once the transfer is complete this connection is closed by the party sending the information. For example if the client is retrieving data from a server, the server will close the connection once all data has been transferred. If the client is the one transferring data to the server, they will terminate the connection. Data connections are opened on a port negotiated by the client and server prior to the command for transferring data. During this negotiation phase, the client will issue either the PORT command for the active mode or the PASV command, for the passive mode. For the active mode, the client issues a `PORT` command to the server signaling that the client will provide an IP and port number to open the data connection back to the client. For the passive mode, the client issues a `PASV` command to indicate that the client will wait for the server to supply an IP and port number, after which the client will create a data connection to the server. The ability to choose between active and passive mode when establishing a data connections is useful for navigating firewalls. Once the IP address and port number have been selected, the party that chose the IP address and port

will begin to listen on the address/port specified and wait for the other party to connect. When the other party connects to the listening party, the data transfer begins. After the data has been transferred, the party that has sent the data will close the Data Connection, signaling end-of-file (EOF).

FTP was one of the first examples to be considered in session type literature, appearing as a theoretical example in one of the seminal papers [71]. Although FTP use is no longer common, it remains an interesting case study for session types, due to the interplay between the control and data connections, with a new data connection being established whenever needed, and disconnected after the transfer takes place.

We limit this case study to the minimum implementation as described in the RFC, made up of the following commands: `USER`, `QUIT`, `PORT`, `TYPE`, `MODE`,`STRU`,`RETR`, `STOR`, `NOOP`, and to these add `PASV` for the passive mode.

We first try to formalise FTP as a protocol between four processes, as described by the RFC, User Protocol Interpreter (UPI), Server Protocol Interpreter (SPI), User Data Transfer Process (UDTP), Server Data Transfer Process (SDTP). SDTP is a data transfer process, which in the "active" mode, establishes the data connection with the "listening" data port. It sets up the parameters for transfer and storage, and transfers data on command from its protocol interpreter, SPI. In the "passive" mode it listens for rather than initiates a connection on the data port. SPI is the server protocol interpreter "listening" for a connection from a UPI and then establishing a control communication connection. It receives standard FTP commands from the UPI, sends replies, and governs the SDTP. UDTP is the data transfer process "listening" on the data port for a connection from a server-FTP process.

UPI is the the user protocol interpreter initiating the control connection from its port to the server-FTP process, initiates FTP commands, and governs the UDTP if that process is part of the file transfer.

In this setting, UDTP and SDTP are dynamically involved in the protocol, only in the commands requiring a data connection. However, in standard MPST, and thus in the standard version of the Scribble tools, a session cannot have dynamic or optional involvement of participants. Furthermore, each involved participant must be present in all choice cases. MPST literature uses work arounds: e.g., adding extra messages, decomposing into separate protocols, session delegation. Instead, we use the Scribble tools as extended by Hu and Yoshida [78], in which they extend multiparty session types with explicit connection actions to support protocols with optional and dynamic participants. Protocol validation is done by a combination of syntactic constraints and explicit error checking, using 1-bounded model checking.

We show an excerpt of the global protocol expressed in Scribble in Listing 8.6 and in Listing 8.7.

In Listing 8.7 the protocol starts with the client's `UPI` connecting to the server's `SPI` and the server authenticating the connection. Upon receiving the username, the server has the choice

to accept the username and request a password, to accept a special username (e.g. anonymous) and continue in anonymous mode, or to send an error message to the client, if for example the username does not exist. After the username has been accepted, the client is then required to send a password, `PASS(str)` or to end the authorization, `QUIT()`. In the first branch of this choice, if the password is accepted, the client has a choice of various commands. The two shown are `PASV` and `PORT` for specifying a data connection mode before connecting the data transfer processes and doing any data transfer by calling the `commands` subprotocol. In Listing 8.6 we show two data commands, `RETR` and `STOR`, and highlight a problem with this represention. Due to the safety requirement that every potentially incoming message in an input choice (i.e., either accept or receive) must be directed from the same role, the highlighted lines cause an error. We can use the usual workaround of adding extra messages between the two roles, `SDTP` and `UDTP`, to pass the Scribble validation. However, this would break interoperability with other implementations. Furthermore, if we want to extend this representation, this version of the Scribble tools can no longer perform validation due to deep recursion and model checking.

```
...
explicit global protocol FTP(role SPI, role UPI, role SDTP, role UDTP){
  connect UPI to SPI;
  _220(str) from SPI to UPI; //server ready
  rec login{ USER(str) from UPI to SPI;
      choice at SPI{
          _331(str) from  SPI to UPI;
          choice at SPI{
              _230(str) from SPI to UPI; //logged in
              choice at UPI{
                  PASV() from UPI to SPI;
                  choice at SPI{ _227(str) from SPI to UPI;
                                 connect SPI to SDTP;
                                 connect UPI to UDTP;
                                 do commands(SPI, UPI, SDTP, UDTP);
                                 disconnect SPI and SDTP;
                                 disconnect UPI and UDTP;}
                               or{...}} //SPI error messages
              or{ PORT() from UPI to SPI;
                  choice at SPI{ _200(str) from SPI to UPI;
                                 connect SPI to SDTP;
                                 connect UPI to UDTP;
                                 do commands(SPI, UPI, SDTP, UDTP);
                                 disconnect SPI and SDTP;
                                 disconnect UPI and UDTP;}
                               or{...}}//SPI error messages
              or{... }} //other commands
          or{   _530(str) from SPI to UPI; //Not logged in.
               continue login; }}
      or{ QUIT() from UPI to SPI;
          _221(str) from SPI to UPI; //Good-bye
          disconnect UPI and SPI;}}
   or{ _530(str) from SPI to UPI; //Not logged in.
       continue login;}}}
```

Listing 8.6: FTP Explicit Global Protocol

```
1  ...
2  aux global protocol commands(role SPI, role UPI, role SDTP, role UDTP){
3  choice at UPI{ RETR(str) from UPI to SPI;
4                 choice at SPI{ _150(str) from SPI to UPI;
5                                command(str) from SPI to SDTP;
6                                command(str) from UPI to UDTP;
7                                transfer(str) connect SDTP to UDTP;
8                                disconnect UDTP and SDTP;
9                                done() from SDTP to SPI;
10                               _226(str) from SPI to UPI;
11                               do dataCommands(SPI, UPI, SDTP, UDTP);
12                           }or{ _421(str) from SPI to UPI;
13                               quit() from SPI to SDTP;
14                               quit() from UPI to UDTP;}
15           }or{ STOR(str) from UPI to SPI;
16                choice at SPI{ _150(str) from SPI to UPI;
17                               command(str) from SPI to SDTP;
18                               command(str) from UPI to UDTP;
19                               transfer(str) connect UDTP to SDTP;
20                               disconnect UDTP and SDTP;
21                               done() from SDTP to SPI;
22                               _226(str) from SPI to UPI;
23                               do dataCommands(SPI, UPI, SDTP, UDTP);
24                           }or{ _421(str) from SPI to UPI;
25                               quit() from SPI to SDTP;
26                               quit() from UPI to UDTP; }
27           }or{ QUIT() from UPI to SPI; _221(str) from SPI to UPI;
28                quit() from SPI to SDTP; quit() from UPI to UDTP; }
```

Listing 8.7: FTP Explicit Global Protocol

We abstract the four different roles from above to two, `Client` and `Server`, represent the protocol in the standard version of Scribble, and perform syntactic based validation. We show an extract of the protocol in Listing 8.6. We show part of the authentication, the commands for the passive and active modes: `PASV()` and `PORT()`, and two data commands for retrieving and storing a file: `RETR(str)` and `STOR(str)`. The following implementation has been tested against dlptest.com[2] and Pure-FTPd[3] on macOS.

[2] https://dlptest.com/ftp-test/
[3] https://www.pureftpd.org/project/pure-ftpd/

```
global protocol FTP(role S, role C) {
  _220(str) from S to C; //server ready
  rec login {
      choice at C {
          USER(str) from C to S;
          choice at S {
              _331(str) from  S to C; //Password required
              choice at C {
                  PASS(str) from C to S;
                  _230(str) from S to C; //logged in
                  rec modes {
                        choice at C { //passive mode
                            PASV() from C to S;
                            choice at S {
                                _227(str) from S to C;
                                choice at C {
                                     STOR(str) from C to S;
                                     choice at S {
                                        _150(str) from S to C;
                                        transfer(str) from C to S;
                                        _226(str) from S to C;
                                        continue modes; }
                                     or {...}}//error replies from S
                                or { RETR(str) from C to S;
                                     choice at S {
                                        _150(str) from S to C;
                                        transfer(str) from C to S;
                                        _226(str) from S to C;
                                        continue modes; }
                                     or {...}}}//error replies from S
                            or {...}}//error replies from S
                        or { PORT() from C to S; //active mode
                            choice at S { _200(str) from S to C;
                                          ... } // client commands
                                     or { ... }} //error replies from S
                        or {...} // other client commands }}
              or { QUIT() from C to S;
                   _221(str) from S to C; }}
          or {...}} // error replies from S
      or {...} // other client commands
      or { QUIT() from C to S;
           _221(str) from S to C; }}}
```

Listing 8.8: FTP: Classic Global Protocol

Using the Scribble tools, we can project the FTP global protocol onto local protocols for the server `S` and the client `C`.

The local protocol for the FTP client `C`, an extract of which is given in Listing 8.9, describes the behaviour of this role. The `_C` in the protocol name indicates that `C` is the local endpoint.

We can now run StMungo on the FTP client protocol Listing 8.9 which produces the following files, where `C` at the beginning of each file name stands for client.

1. `CProtocol.protocol`: the typestate specification representing the FTP client's local protocol. The send and receive operations are translated as Java methods (Listing 8.10).

2. `CRole.java`: the Java API implementing the FTP client. This class implements the typestate `CProtocol` over Java sockets (Listing 8.11).

3. `CMain.java`: an optional file, that gives a minimum logic for the client `CRole` and provides a `main()` method (Listing 8.12).

```
local protocol FTP_C(role S, role C) {
  _220(str) from S; //server ready
  rec login {
    choice at C {
        USER(str) to S;
        choice at S {
            _331(str) from  S; //Password required
            choice at C {
                PASS(str) to S;
                _230(str) from S; //logged in
                rec modes {
                      choice at C { //passive mode
                          PASV() to S;
                          choice at S {
                              _227(str) from S;
                              choice at C {
                                   STOR(str) to S;
                                   choice at S {
                                      _150(str) from S;
                                      transfer(str) to S;
                                      _226(str) from S;
                                      continue modes; }
                                   or {...}}//error replies from S
                              or { RETR(str) to S;
                                   choice at S {
                                      _150(str) from S;
                                      transfer(str) to S;
                                      _226(str) from S;
                                      continue modes; }
                                   or {...}}}//error replies from S
                          or {...}}//error replies from S
                      or { PORT() to S; //active mode
                          choice at S { _200(str) from S;
                                        ... } // client commands
                                   or { ... }} //error replies from S
                      or {...} // other client commands }}
            or { QUIT() to S;
                 _221(str) from S; }}
        or {...}} // error replies from S
    or {...} // other client commands
    or { QUIT() to S;
         _221(str) from S; }}}
```

Listing 8.9: FTP: Client Local Protocol

```
typestate CProtocol {
  State0  = { String receive_220StrFromS(): State1 }
  State1  = { void send_USERToS(): State2,
              void send_QUITToS(): State80 }
  State2  = { void send_USERStrToS(String): State3 }
  State3  = { CChoice1 receive_CChoice1LabelFromS(): <_331: State4, ... > }
  State4  = { String receive_331StrFromS(): State5 }
  State5  = { void send_PASSToS(): State6,
              void send_QUITToS(): State76 }
  State6  = { void send_PASSStrToS(String): State7 }
  State7  = { String receive_230StrFromS(): State8 }
  State8  = { void send_PASVToS(): State9,
              void send_PORTToS(): State31,
              void send_QUITToS(): State52,
              // ... methods implementing other commands  }
  State9  = { void send_PASVToS(): State10 }
  State10 = { CChoice2 receive_CChoice2LabelFromS(): <_227: State11, ... > }
  State11 = { String receive_227StrFromS(): State12 }
  State12 = { void send_STORToS(): State13,
              void send_RETRToS(): State21 }
  State13 = { void send_STORStrToS(String): State14 }
  State14 = { CChoice3 receive_CChoice3LabelFromS(): <_150: State15, ... > }
  State15 = { String receive_150StrFromS(): State16 }
  State16 = { void send_transferStrToS(String): State17 }
  State17 = { String receive_226StrFromS(): State8 }
  ...
  State21 = { void send_RETRStrToS(String): State22 }
  State22 = { CChoice4 receive_CChoice4LabelFromS(): <_150: State23, ... >}
  State23 = { String receive_150StrFromS(): State24 }
  State24 = { void send_transferStrToS(String): State25 }
  State25 = { String receive_226StrFromS(): State8 }
  ...
  State31 = { void send_PORTToS(): State32 }
  State32 = { CChoice5 receive_CChoice5LabelFromS(): <_200: State33, ...>}
  State33 = { String receive_200StrFromS(): State34 }
  ...
  State76 = { void send_QUITToS(): State77 }
  ...
  State80 = { void send_QUITToS(): State81 }
  State81 = { String receive_221StrFromS(): end} }
```

Listing 8.10: FTP Client Typestate Specification

```
...
@Typestate("CProtocol")
public class CRole{
  public CRole() { ...//Bind the sockets and accept a connection
    try { // Create the read and write streams
        socketSIn = new BufferedReader(...);
        socketSOut = new PrintWriter(...);}
    catch (IOException e) {...}}
  public String receive_220StrFromS() {
    String line = "";
    try { line = this.socketSIn.readLine(); }
    catch(IOException e) {...}
    return line;}
  public void send_USERToS() { this.socketSOut.println("USER"); }
  public void send_USERStrToS(String payload0) {
    this.socketSOut.println(payload0); }
  public CChoice1 receive_CChoice1LabelFromS() {
    String stringLabelCChoice1 = "";
    try { stringLabelCChoice1 = this.socketSIn.readLine(); }
    catch(IOException e) {...}
    switch(stringLabelCChoice1) {
      case "_331":
        return CChoice1._331;
        ... }}
  public String receive_331StrFromS() {
    String line = "";
    try { line = this.socketSIn.readLine(); }
    catch(IOException e) {...}
    return line; }
  public void send_PASSToS() { this.socketSOut.print("PASS"); }
  public void send_PASSStrToS(String payload0) { this.socketSOut.println(payload0);
    }
  public String receive_230StrFromS() {
    String line = "";
    try { line = this.socketSIn.readLine(); }
    catch(IOException e) {...}
    return line; }
  public void send_PASVToS() { this.socketSOut.println("PASV"); }
  public void send_PORTToS() { this.socketSOut.println("PORT"); }
  ...  }
```

Listing 8.11: FTP: Client Role

```
public static void main(String[] args) {
  CRole currentC =  new CRole();
  System.out.println("Received: " + currentC.receive_220StrFromS());
  _login: do{ String sread1 = safeRead(readerC); //input command
    switch(sread1){
      case "USER":
        currentC.send_USERToS();
        currentC.send_USERStrToS(safeRead(readerC));
        switch(currentC.receive_CChoice1LabelFromS()) {
            case _331:
              System.out.println("Received:"+currentC.receive_331StrFromS());
              String sread2 = safeRead(readerC); //input command
              switch(sread2){
                case "PASS":
                  currentC.send_PASSToS();
                  String payload7 = safeRead(readerC);
                  currentC.send_PASSStrToS(payload7);
                  System.out.println("Received: " + currentC.receive_230StrFromS());
                  _modes: do{ String sread3 = safeRead(readerC); //input command
                    switch(sread3){
                      case "PASV":
                        currentC.send_PASVToS();
                        switch(currentC.receive_CChoice2LabelFromS()) {
                          case _227:
                            System.out.println(currentC.receive_227StrFromS());
                            String sread4 = safeRead(readerC); //input command
                            switch(sread4){
                              case "STOR":
                                currentC.send_STORToS();
                                currentC.send_STORStrToS(safeRead(readerC));
                                switch(currentC.receive_CChoice3LabelFromS()) {
                                  case _150:
                                    System.out.println(currentC.receive_150StrFromS());
                                    String payload18 = safeRead(readerC);
                                    currentC.send_transferStrToS(payload18);
                                    System.out.println(currentC.receive_226StrFromS());
                                    continue _modes;
                                  /* error cases */  } break _modes;
                              case "RETR":
                                currentC.send_RETRToS();
                                String payload28 = safeRead(readerC); //input filename
                                currentC.send_RETRStrToS(payload28);
                                switch(currentC.receive_CChoice4LabelFromS()) {
                                  case _150:
                                    String payload30 = currentC.receive_150StrFromS();
                                    System.out.println("Received: " + payload30);
                                    currentC.send_transferStrToS(safeRead(readerC));
                                    String payload34 = currentC.receive_226StrFromS();
                                    System.out.println("Received: " + payload34);
                                    continue _modes;
                                    /* error cases */} break _modes; } break _modes;
          /* other FTP commands */ ...   }while(true);}
```

Listing 8.12: FTP: Client Main

value

8.4 Paxos

Paxos is a family of protocols for reaching consensus in a network that operates under conditions of unreliability. In this section we look at the basic Paxos protocol as described in [88, 89], and forms the basis of many more advanced consensus protocols. It ensures that network agents can agree on a single value in the presence of failures. Paxos is the first consensus algorithm that has been formally proven to be correct. Coordination and consensus play an important role in data center and cloud computing, particularly in leader election, group membership, resource management, or consistent replication of nodes. Despite being widely used, Paxos is notoriously difficult to understand, and real-world implementations have brought forth many problems that are not taken into account by the theoretical model of Paxos [28].

This algorithm defines a peer-to-peer protocol for selecting a single value from the values that are proposed and informs all participants what that is. It is based on a majority (or quorum) rule to ensure that only one value is agreed upon. There is no concept of a dedicated leader in Paxos, any node may propose a value and attempt to achieve resolution, at the same time as others, which may result in overriding the efforts of other nodes. Despite this, the algorithm ensures safety thgrough the use of ids for each proposed value, and the concept of a quorum. Eventually, a majority of nodes will agree upon a proposed value, which will become the final chosen value. The basic algorithm can be extended and generalised to obtain more complex protocols, such as Multi-Paxos, an extension of the basic protocol for running efficiently with the same proposer for multiple rounds.

A correct implementation of the protocol ensures that: only a value that has been proposed for consensus is chosen; the nodes within the network agree on a single value; and that a node is never wrongly informed that a value is chosen for consensus.

The Paxos setting assumes asynchronous non-Byzantine communication that operates under the following assumptions: *messages* can take arbitrarily long to be delivered, can be duplicated, but are not delivered corrupted; *agents* operate at an arbitrary speed, may stop operating and may restart. However, it is assumed that agents maintain persistent storage that survives crashes.

Paxos agents implement three roles: i) a *proposer* agent proposes vs towards the network for reaching consensus; ii) an *acceptor* accepts a value from those proposed, whereas a majority of acceptors accepting the same value implies consensus and signifies protocol termination; and iii) a *learner* discovers the chosen consensus value. Each Paxos node can act as any or all 3 roles. The network agents act autonomously and propose values for consensus to the other agents within the network. If eventually a majority of agents run for long enough without failing, consensus on one of the proposed values is guaranteed.

A run of the protocol may proceed over several rounds. A successful round has two phases:

prepare during which a proposer checks with the acceptors whether any of them have already received a proposal, if so it will propose an existing value, if not it will propose its own value; and *accept* during which if a majority of acceptors agree to this value then that is our consensus. The check during the prepare phase ensures that in the case where a value v has already been chosen by a majority of acceptors, broadcasting a new proposal request with a higher proposal id will result in choosing the already chosen consensus value v.

A session type representation of Paxos is used to check that implementations correctly follow the protocol. Session types can also help to identify subtle interactions such as branching or dropping sessions. Furthermore, a session type representation allows for the basic algorithm to be easily extended while still providing formal guarantees.

Implementation via [St]Mungo

We first attempt to implement the protocol via Scribble and [St]Mungo. However, in trying to do so, a few shortcomings have become apparent, such as representing broadcasting; representing a quorum; expressing the dynamic aspects such as processes failing, or restarting and then rejoining; or representing a collection of agents. We then limit this implementation to a minimal Paxos setting: a proposer and three acceptors. More agents can be added similarly if needed, however most consensus clusters tend to be rather small, three to five nodes. The resulting Java implementation can be used to guide the programmer in implementing a full Paxos system.

We start by giving the Scribble global protocol for Paxos in Listing 8.13. The protocol starts with the client, `C`, sending a `request` to the proposer, `P`, for a value, `v`, to be proposed. The proposer then sends a `prepare` message carrying a unique `ID`. Each of the three acceptors has a choice to either send a `promise` message that they will not accept any request with a smaller `ID` than the current one, or send a `reject` message. The `promise` message also carries with it value `v`, which can either be null, or carry the value that was previously agreed upon.

If the proposer receives promises from a majority of acceptors, it will send an `acceptV` message to the acceptors, carrying two payloads, the proposal `ID` and the value `v`. Each acceptor has a choice to either send an `accepted` message or to `reject` the value. If the proposer receives `accepted` from a majority of acceptors, consensus has been reached, and the participants can be notified. If not it will restart the protocol and attempt to propose its value with a higher ID.

```
1  global protocol Paxos(role P, role A1, role A2, role A3, role C) {
2    request(v) from C to P;
3    rec X{ //Prepare Phase
4        prepare(ID) from P to A1;
5        prepare(ID) from P to A2;
6        prepare(ID) from P to A3;
7        choice at A1{ promise(ID, v) from A1 to P; }
8                  or{ reject() from A1 to P; }
9        choice at A2{ promise(ID, v) from A2 to P; }
10                 or{ reject() from A2 to P; }
11       choice at A3{ promise(ID, v) from A3 to P; }
12                 or{ reject() from A3 to P; }
13       //Accept Phase
14       choice at P{ acceptV(ID, v) from P to A1;
15                    acceptV(ID, v) from P to A2;
16                    acceptV(ID, v) from P to A3;
17                    choice at A1{ accepted(ID, v) from A1 to P; }
18                              or{ reject() from A1 to P; }
19                    choice at A2{ accepted(ID, v) from A2 to P; }
20                              or{ reject() from A2 to P; }
21                    choice at A3{ accepted(ID, v) from A3 to P; }
22                              or{ reject() from A3 to P; }
23                    choice at P{ notify(v) from P to A1;
24                                 notify(v) from P to A2;
25                                 notify(v) from P to A3; }
26                             or{ restart() from P to A1;
27                                 restart() from P to A2;
28                                 restart() from P to A3;
29                                 continue X; } }
30                 or{ restart() from P to A1;
31                     restart() from P to A2;
32                     restart() from P to A3;
33                     continue X; } }
34   response(v) from P to C; }
```

Listing 8.13: Paxos: Global Protocol

Using the Scribble tools we can obtain the local protocols for each of the roles. Here, we show only the local protocol for the proposer, capturing the interactions of role `P`, in Listing 8.14.

```
local protocol Paxos_P(role P, role A1, role A2, role A3, role C) {
    request(v) from C;
    rec X {
        prepare(ID) to A1;
        prepare(ID) to A2;
        prepare(ID) to A3;
        choice at A1 { promise(ID, v) from A1; }
                  or { reject() from A1; }
        choice at A2 { promise(ID, v) from A2;
                  or { reject() from A2; }
        choice at A3 { promise(ID, v) from A3; }
                  or { reject() from A3; }
        choice at P { acceptV(ID, v) to A1;
                      acceptV(ID, v) to A2;
                      acceptV(ID, v) to A3;
                      choice at A1 { accepted(ID, v) from A1; }
                                or { reject() from A1; }
                      choice at A2 { accepted(ID, v) from A2; }
                                or { reject() from A2; }
                      choice at A3 { accepted(ID, v) from A3; }
                                or { reject() from A3; }
                      choice at P { notify(v) to A1;
                                    notify(v) to A2;
                                    notify(v) to A3; }
                               or { restart() to A1;
                                    restart() to A2;
                                    restart() to A3;
                                    continue X; }}
                 or { restart() to A1;
                      restart() to A2;
                      restart() to A3;
                      continue X; }}
    response(v) to C; }
```

Listing 8.14: Paxos: Proposer Protocol

Using StMungo, we can obtain the typestate, and the corresponding APIs and a skeleton implementation for each endpoint. We show the typestate for the proposer, `P`, is given in Listing 8.15. Once we extend it with the necessary logic for inspecting promises, or selecting a value, this implementation allows us to run a very basic Paxos system, which can then be extended with any additional functionality needed.

```
typestate PProtocol {
    State0 = {String receive_requestvFromClient(): State1}
    State1 = {void send_prepareIDToA1(String): State2}
    State2 = {void send_prepareIDToA2(String): State3}
    State3 = {void send_prepareIDToA3(String): State4}
    State4 = {PChoice1 receive_PChoice1LabelFromA1():
              <PROMISE: State5, REJECT: State7>}
    State5 = {String receive_promiseIDFromA1(): State6}
    State6 = {String receive_promisevFromA1(): State7}
    State7 = {void receive_rejectFromA1(): State8}
    State8 = {PChoice1 receive_PChoice1LabelFromA2():
              <PROMISE: State9, REJECT: State11>}
    State9 = {String receive_promiseIDFromA2(): State10 }
    State10 = {String receive_promisevFromA2(): State11 }
    State11 = {void receive_rejectFromA2(): State12}
    State12 = {PChoice1 receive_PChoice1LabelFromA3():
               <PROMISE: State13, REJECT: State15>}
    State13 = {String receive_promiseIDFromA3(): State14}
    State14 = {String receive_promisevFromA3(): State15}
    State15 = {void receive_rejectFromA3(): State16}
    State16 = {void send_ACCEPTVToA1(): State17, void send_RESTARTToA1():
     State39}
    State17 = {void send_acceptVIDvToA1(String,String): State18}
    State19 = {void send_acceptVIDvToA3(String,String): State20}
    State20 = {PChoice2 receive_PChoice2LabelFromA1():
               <ACCEPTED: State21, REJECT: State23>}
    State21 = {String receive_acceptedIDFromA1(): State22}
    State22 = {String receive_acceptedvFromA1(): State23}
    State23 = {void receive_rejectFromA1(): State24}
    State24 = {PChoice2 receive_PChoice2LabelFromA2():
               <ACCEPTED: State25, REJECT: State27>}
    ...
    State32 = {void send_NOTIFYToA1(): State33,
               void send_RESTARTToA1(): State36}
    State33 = {void send_notifyvToA1(String): State34}
    State34 = {void send_notifyvToA2(String): State35}
    State35 = {void send_notifyvToA3(String): State42}
    State36 = {void send_restartToA1(): State37}
    ...
    State41 = {void send_restartToA3(): State1}
    State42 = {void send_responsevToClient(String): end}}
```

Listing 8.15: Paxos: Proposer Typestate

In the next paragraphs we present a different approach at implementing the protocol, capturing node failure, a dynamic number of agents, and concurrent proposals. We obtain this more comprehensive representation of Paxos by using the session type system for unreliable broadcast

$$
\begin{aligned}
\mathsf{PaxosType} &= !\texttt{prepare}.?\texttt{promise}.\oplus\left\{\begin{array}{l}\mathsf{accept}:\ !(\texttt{round},\texttt{value}).?(\texttt{round},\texttt{value}).\oplus\left\{\begin{array}{l}\mathsf{restart}:\ \texttt{end},\\ \mathsf{chosen}:\ \texttt{end}\end{array}\right\}\\ \mathsf{restart}:\ \texttt{end}\end{array}\right\}\\
\mathsf{PaxosNode}^{\mathsf{id}}_{r,v} &= [\mathsf{Paxos}^{\mathsf{id}}_{r,v}]\\
\mathsf{Paxos}^{\mathsf{id}}_{r,v} &= \mathsf{def}\\
&\qquad \mathsf{Proposer}(x,y) \stackrel{\mathsf{def}}{=} a_!(s).\breve{s}!\langle x\rangle.\breve{s}?(\{(r_i,v_i)\}_{i\in I}).\\
&\qquad\qquad \textbf{if } |I| > \tfrac{M}{2} \textbf{ then}\\
&\qquad\qquad\quad \breve{s}\triangleleft\mathsf{accept}.\breve{s}!\langle(x,v=\mathsf{choose}(\{(r_i,v_i)\}_{i\in I},\mathsf{id}))\rangle.\\
&\qquad\qquad\quad \breve{s}?(\{(r_i,v_i)\}_{i\in J}).\ \textbf{if } |J| > \tfrac{M}{2} \textbf{ then}\\
&\qquad\qquad\qquad\qquad \breve{s}\triangleleft\mathsf{chosen}.\mathsf{Paxos}\langle x,y\rangle\\
&\qquad\qquad\qquad\quad \textbf{else}\\
&\qquad\qquad\qquad\qquad \breve{s}\triangleleft\mathsf{restart}.\mathsf{Paxos}\langle x,y\rangle\\
&\qquad\qquad \textbf{else}\\
&\qquad\qquad\quad \breve{s}\triangleleft\mathsf{restart}.\mathsf{Paxos}\langle x,y\rangle\\
&\qquad \mathsf{Acceptor}(x,y) \stackrel{\mathsf{def}}{=} a_?(s).s?(x').\\
&\qquad\qquad \textbf{if } (x' > x) \textbf{ then}\\
&\qquad\qquad\quad s!\langle x,y\rangle.s\triangleright\left\{\begin{array}{l}\mathsf{accept}:\ s?(x',y').s!\langle x',y'\rangle.\\ \qquad\qquad s\triangleright\left\{\begin{array}{l}\mathsf{chosen}:\ \mathsf{Paxos}\langle x',y'\rangle\\ \mathsf{restart}:\ \mathsf{Paxos}\langle x,y\rangle,\end{array}\right\}\\ \mathsf{restart}:\ \mathsf{Paxos}\langle x,y\rangle\end{array}\right\}\\
&\qquad\qquad \textbf{else}\\
&\qquad\qquad\quad \mathsf{Paxos}\langle x,y\rangle\\
&\qquad \mathsf{Paxos}(x,y) \stackrel{\mathsf{def}}{=} \mathsf{Proposer}\langle x+1,y\rangle + (\mathsf{Acceptor}\langle x,y\rangle \diamond \mathsf{Paxos}(x,y))\\
&\ \mathsf{in}\\
&\ \ \mathsf{Paxos}\langle r,v\rangle
\end{aligned}
$$

Figure 8.2: Implementation of the Paxos consensus protocol

communication devised by Gutkovas, Kouzapas and Gay.

Implementation in the Unreliable Broadcast Session Communication System

We present an implementation of the Paxos consensus protocol using the unreliable broadcast session communication system introduced by Kouzapas *et al.* [87]. Inspired by the practice of ad-hoc and wireless sensor network, the semantics of the calculus have the necessary mechanisms to support safe session interaction and recovery. The system ensures asynchronous, non-Byzantine communication, that messages are not corrupted nor duplicated, and that agents may operate at arbitrary speeds.

Prior to the example, we give a quick informal presentation of the syntax of the calculus. The

calculus defines the syntax for *network nodes*. Specifically, a network node,

$$N = [\,P \mid \prod_{i\in I} s_i[c,\tilde{m}_i]\,]$$

composes a binary session π-calculus process P with a finite parallel composition of session buffer terms, $\prod_{i\in I} s_i[c,\tilde{m}_i]$. A buffer term, $s[c,\tilde{m}]$, represents a first-in first-out message buffer that interacts on session endpoint s. The buffer stores messages $\tilde{m}$ and keeps track of the session endpoint state using integer counter c. Multiple network nodes can be composed in parallel $N_1|\ldots|N_n$ to form a network. The term, $\prod_{j\in J}[\,P_j \mid \prod_{i\in I_j} s_i[c,\tilde{m}_i]\,]$ is used to represent a parallel composition of network nodes.

The main session communication operations include asynchronous broadcast and asynchronous gather in the presence of link failure and message loss. The interaction within a session name s is defined between an $\check{s}$-endpoint, uniquely used by a single network node, and a s-endpoint shared by an arbitrary number of network nodes. The one to many correspondence between endpoints gives rise to the broadcasting operation, where the $\check{s}$-endpoint broadcasts a value towards the s-endpoints, and the gather operation, where the $\check{s}$-endpoint gathers messages sent from the s-endpoints. The rest of the syntax of session types follows the syntax of standard binary session types.

Figure 8.2 describes the implementation of the Paxos protocol in this framework. Similar to the previous implementation, we assume for simplicity that a learner has the same implementation as a proposer. The implementation assumes that expressions contain finite sets of integer tuples, which we write as: $\{(r_i,v_i)_i\}_{i\in I}$.

The interaction for establishing a consensus value takes place within a single session involving the network nodes. The communication behaviour of an acceptor is described by the session type $\mathsf{PaxosType}$, whereas that of a proposer is described by the dual type $\overline{\mathsf{PaxosType}}$.

A Paxos agent is described by network node $\mathsf{PaxosNode}^{\mathsf{id}}_{r,v} = [\,\mathsf{Paxos}^{\mathsf{id}}_{r,v}\,]$ where r is the number of the current proposal id, v is the proposed value that corresponds to proposal id r, and id is a unique node identity number. A Paxos agent non-deterministically behaves either as a proposer: $\mathsf{Proposer}(x,y)$ or as an acceptor: $\mathsf{Acceptor}(x,y) \diamond \mathsf{Paxos}(x,y)$.

During the computation a Paxos agent may restart; to do this it terminates its current sessions and proceeds to the initial Paxos network, $\mathsf{Paxos}_{x,y}$. Note that each time an initial Paxos agent enters a new protocol run it establishes a new session.

If a Paxos agent decides to act as a proposer, it does so by increasing its current proposal id and proceeding to process $\mathsf{Proposer}\langle r+1,v\rangle$. It then requests a new session and enters the Prepare phase. All the Paxos agents that accept a session request act as acceptors. The proposer then broadcasts towards the network a *prepare* message request, type `prepare`, that contains the proposal id r.

All acceptors that receive the *prepare* message check whether the proposal id is greater than the one they currently have. If it is not, they drop the session and restart the computation

proceeding to process $\mathsf{Paxos}\langle r, v\rangle$. Otherwise, they reply with a *promise* message, type `promise`, not to respond to a prepare message with a lower round number. The promise message contains the current proposal id and the current consensus value of the acceptor. If this is the first time the acceptor is involved in a consensus round then the promise message will contain empty values for the two payloads $(\varepsilon, \varepsilon)$. Here we assume that for all proposal ids r it holds that $r > \varepsilon$.

After a majority of acceptors reply with a promise message, the protocol enters the Accept phase. The proposer gathers all the promises as a set of messages, $\{(r_i, v_i)\}_{i\in I}$, and then checks whether the majority of acceptors have replied using condition $|I| \leq \frac{M}{2}$, with M being the number of the nodes in the network. Note that in the Proposer agent we use the set notation $\{(r_i, v_i)\}_{i\in I}$ in place of variable in an input process. If the check fails the proposer sends a restart label to all the acceptors, and restarts its own computation by proceeding to process $\mathsf{Paxos}\langle r, v\rangle$. All acceptors that receive label restart also restart their computation by proceeding to process $\mathsf{Paxos}\langle r, v\rangle$.

If the majority check is passed, the proposer selects a value to submit to the acceptors by inspecting the promises received and choosing the value corresponding to the highest proposal id received in a promise message. If no value is received then it will use its own value. This is expressed bellow by:

$$\begin{aligned} v_k &= \mathsf{choose}(\{(r_i, v_i)\}_{i\in I}, \mathsf{id}) \text{ when } \forall i \in I, r_k \geq r_i \qquad \text{and} \\ \mathsf{id} &= \mathsf{choose}(\{(r_i, v_i)\}_{i\in I}, \mathsf{id}) \text{ when } \forall i \in I, r_i = \varepsilon \end{aligned}$$

The proposer then broadcasts an *accept* message containing an accept label followed by a tuple of the current proposal id and the chosen value. The acceptors reply with a message of type (`round`, `value`), containing their current round number. These messages are gathered by the proposer and checked for majority, in which case consensus has been reached and the acceptors will be informed via label chosen. The proposer will then proceed to process $\mathsf{Paxos}\langle r, v\rangle$. All acceptors that receive the *chosen* message update their proposal id and their consensus value, and proceed to state $\mathsf{Paxos}\langle r, v\rangle$.

If lack of majority is detected, the proposer restarts all agents within the session via label restart, and increments its proposal id for the next round.

Network node $\mathsf{PaxosNode}_{n,v}$ can be typed using the following typing judgement:

$$a : \mathsf{PaxosType}; \varnothing \vdash \mathsf{PaxosNode}^{\mathsf{id}}_{n,v}$$

Shared channel a uses type PaxosType, thus all established sessions of the computation follow the behaviour of the protocol as described by the PaxosType session type. Subsequently, a network that describes a set of nodes that run the protocol is defined as:

$$N = \prod_{i\in I} \mathsf{PaxosNode}^{i}_{\varepsilon,\varepsilon}$$

Network N is typed using typing judgement $a : \mathsf{PaxosType}; \varnothing \vdash N$.

More complicated Paxos networks can be described. For example, we can allow a Paxos

agent that acts as an acceptor to establish multiple sessions with different proposers during an execution and explicitly drop the session with the lowest proposal id:

$$
\begin{array}{lcl}
\mathsf{Acceptor}(x,y) & \stackrel{\mathsf{def}}{=} & a_?(s).s?(x').\mathsf{Acc}\langle s,x,x',y\rangle \\
\mathsf{Acc}(w,x,x',y) & \stackrel{\mathsf{def}}{=} & \textbf{if } (x' > x) \textbf{ then} \\
 & & \quad w!\langle x,y\rangle. \\
 & & \qquad (\mathsf{AcceptPhase}\langle w,x,y\rangle \\
 & & \quad + \quad a_?(s').s?(x''). \\
 & & \qquad \textbf{if } (x'' > x') \\
 & & \qquad\quad \mathsf{Acc}\langle s,x,x'',y\rangle \\
 & & \qquad \textbf{else} \\
 & & \qquad\quad \mathsf{AcceptPhase}\langle w,x,y\rangle) \\
 & & \textbf{else} \\
 & & \quad \mathsf{Paxos}\langle x,y\rangle \\
\mathsf{AcceptPhase}(w,x,y) & \stackrel{\mathsf{def}}{=} & w \triangleright \left\{ \begin{array}{ll} \mathsf{accept}: & w?(x',y').\mathsf{Paxos}\langle x',y'\rangle, \\ \mathsf{restart}: & \mathsf{Paxos}\langle x,y\rangle \end{array} \right\}
\end{array}
$$

The definition of the acceptor has the option, expressed as a non-deterministic choice $+$, to establish a new session, i.e. enter a prepare phase with a different proposer, while in the accept phase of another proposal. The proposal ids from the two sessions are compared, and based on this the session with the lowest id will be dropped. The computation will proceed accordinbg to the type of the session with the higher proposal id.

This representation ensures that the interaction takes place within a session as defined by the session type. This lifts the burden from the programmer to check for deadlocks and type mismatches, and leaves only the burden for implementing correctly the algorithmic logic.

Part IV

Conclusion

Chapter 9

Conclusion

Type-theoretic techniques can be used to address many of the issues that come with communication centred systems. These systems are designed around the idea of communication between components, and they can be complex to design and implement due to the need to coordinate the interactions between these components. Formally describing the communication between the components of a system allows it to be verified by static typechecking. This provides lightweight verification that can catch errors such as the interactions that occur do not follow the prescribed protocol.

9.1 Research Questions Revisited

To summarise the work in this thesis we look back at the two research questions set within the first chapter, in Section 1.1.

9.2 Research Questions

Q1 *What is the relationship between session types and linearity or affinity, and how can we check resource sharing and aliasing to guarantee type safety?*

Resource sharing and aliasing are two common sources of errors in concurrent programming such as race conditions or unexpected behaviour. To ensure type safety session type systems usually enforce linearity and affinity properties to prevent resource sharing and aliasing at compile-time. We have presented a new system of multiparty session types with capabilities, which allows sharing of resources in a way that generalises the strictly linear or affine access control typical of session type systems. The key technical idea is to separate a channel from the capability of using the channel. This allows channels to be shared, while capabilities are linearly controlled. We use a form of existential typing to maintain the link between a channel and its capability, while both are transmitted in messages. We have

proved communication safety, formulated as a subject reduction theorem.

Q2 *How can session types be adapted to support real-world case studies, and can we assess if they are really beneficial?* Session types can be adapted and extended to handle more complex communication patterns and protocols. This involves developing and extending session type tools to capture the requirements of specific applications and ensure that they integrate well with the types already present in the language, in this case Java. Chapter 7 has given an overview of the [St]Mungo toolchain, its usage and extensions made to it to better support session types in practice. In Chapter 8 we have presented several cases studies showing how session types can be successfully used in practice.

9.3 Future Work

The area of session types has a huge scope for further research. We will discuss some potential directions for the work presented in this thesis.

Multiparty session types with capabilities. In Part II we have considered multiparty session types extended with capabilities to allow resource sharing. An area of future work is to prove progress and deadlock-freedom properties for interleaving sessions along the lines of Coppo *et al.* [35]. Another possibility is to apply our techniques to functional languages with session types such as the one in [63].

Another interesting line of future work is to explore how this could be applied in the context of CHERI (Capability Hardware Enhanced RISC Instructions) [141]. CHERI is a is a hybrid capability architecture aimed at improving security in computer systems by using hardware capabilities. Capabilities can be thought of as a type of token that represents a permission to access a resource, and can be used to provide fine-grained access control to system resources. A capability can specify access permissions such as read, write, execute, or a combination of these, as well as other metadata such as the size of the memory block. By using capabilities, CHERI can provide memory protection, prevent unauthorized access or modification of memory. For example, if a capability only allows read access to a memory address, any attempt to write to that address will be prevented. In addition, CHERI capabilities can enforce spatial memory safety, preventing buffer overflows or other memory-related security vulnerabilities. The type system presented in Part II could be extended based on hardware capabilities to provide enforceable specifications for software running on CHERI hardware that would offer stronger guarantees its behaviour.

Typestate Programming in Java. An area of future work is to evaluate the usability of the [St]Mungo/Mungo toolchain and compare it to other available tools. New programming language constructs are more often than not introduced without thorough exploration of their suitability for

their intended purpose or practical application. While proving they solve the problem is a good thing, it is equally important to evaluate their effectiveness and usability in real-world scenarios.

Session types have been developed for some time now with industry input, and a closer look at their effect on software development is in order. Otherwise, we run the risk of developing something that may not be quite suitable. An example of this can be seen in gradual typing, another very active area of research, which is now having its practicality called into question [129]. By identifying which designs and implementations help or hinder programmers, we can improve them to help developers use session type effectively.

Assessing the effectiveness of session types in real-world case studies requires evaluating their impact on software development practices, as well as their ability to detect and prevent errors. This can be achieved through empirical studies that measure the performance and productivity of developers who use session types compared to those who do not. Additionally, long term industry case studies can be used to assess the impact of session types on the reliability and robustness of software systems.

There are different approaches to adding session types to existing languages that can be used to assess the effectiveness of session types. In the case of Java, we have Session J [79], Mungo, and Hu and Yoshida's API generation [77]. However, there has not been a comparison of the benefits of each approach for practical programming. Most papers on adding session types to programming languages focus on the language, not on the associated development environment or methodology. Hu's API generation approach allows a standard Java IDE such as Eclipse to inform the programmer about protocol errors. There are additional ways in which an IDE could help programmers, for example by showing a transition diagram or the complete session type. By evaluating the impact of session types on software development practices and the reliability of software systems, we can gain a better understanding of their value in real-world settings.

Appendix A

Proofs for Resource Sharing via Capability-Based Multiparty Session Types

A.1 Proofs

Proposition A.1.1. $\mathtt{unf}\left(\overline{H}\right) = \overline{\mathtt{unf}(H)}$

Proof. By induction on the unfolding of H. [123] □

Proposition A.1.2. *Let S, S' be closed session types. If $S \leqslant S'$, then for all $\mathbf{p}$ also $S \upharpoonright \mathbf{p} \leqslant S' \upharpoonright \mathbf{p}$.*

Proof. Standard, can be found in [123]. □

Proposition A.1.3. *For all session types S and roles $\mathbf{p}$, $\mathtt{unf}(S) \upharpoonright \mathbf{p} = \mathtt{unf}(S \upharpoonright \mathbf{p})$.*

Proof. By induction on the unfolding of S. [123] □

Proposition A.1.4. *If $(\Gamma, C) \longrightarrow^* (\Gamma', C')$ then $\mathrm{dom}(\Gamma) = \mathrm{dom}(\Gamma')$ and $\mathrm{dom}(C) = \mathrm{dom}(C')$.*

Proof. We first verify the following, by induction on the size of $\mathrm{dom}(\Gamma)$: $(\Gamma, C) \longrightarrow (\Gamma', C')$ implies $\mathrm{dom}(\Gamma) = \mathrm{dom}(\Gamma')$ and $\mathrm{dom}(C) = \mathrm{dom}(C')$.

Then, we can prove the main statement, by induction on the length of the sequence of reductions in $(\Gamma, C) \longrightarrow^* (\Gamma', C')$. The base case is trivial, with 0 reductions, and $(\Gamma, C) = (\Gamma', C')$, while in the inductive case, we apply the induction hypothesis. □

Proposition A.1.5. *If $(\Gamma, x : U; C)$ is consistent, then $(\Gamma; C)$ is consistent.*

Proof. The proof is straightforward, by noticing that consistency (Definition 4.5.1) does not depend on $x : U$. □

Proposition A.1.6. *If $(\Gamma, \mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho_{\mathbf{p}}), \rho_{\mathbf{p}} : \{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}\}; C \otimes \{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}\})$ is consistent, then $(\Gamma; C)$ is consistent.*

Proof. Assume that $(\Gamma, \mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho_\mathbf{p}), \rho_\mathbf{p} : \{\rho_\mathbf{p} \mapsto S_\mathbf{p}\}; C \otimes \{\rho_\mathbf{p} \mapsto S_\mathbf{p}\})$ is consistent. By Definition 4.5.1 $\forall \mathtt{s}[\mathbf{q}], \rho_\mathbf{q}, \mathtt{s}[\mathbf{r}], \rho_\mathbf{r} \in \mathrm{dom}\,(\Gamma, \mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho_\mathbf{p}), \rho_\mathbf{p} : \{\rho_\mathbf{p} \mapsto S_\mathbf{p}\}, \mathtt{s}[\mathbf{r}] : \mathtt{tr}(\rho_\mathbf{r}), \rho_\mathbf{r} : \{\rho_\mathbf{r} \mapsto S_\mathbf{r}\})$ with $\mathbf{q} \neq \mathbf{r}$ implies $\overline{\Gamma(\rho_\mathbf{q}) \upharpoonright \mathbf{r}} \leqslant \Gamma(\rho_\mathbf{r}) \upharpoonright \mathbf{q}$. Since $\mathrm{dom}\,(\Gamma) = \mathrm{dom}\,(\Gamma, \mathtt{s}[\mathbf{p}], \rho_\mathbf{p}) \setminus \{\mathtt{s}[\mathbf{p}], \rho_\mathbf{p}\}$, we also have $\forall \mathtt{s}[\mathbf{q}], \mathtt{s}[\mathbf{r}] \in \mathrm{dom}\,(\Gamma)$ with $\mathbf{q} \neq \mathbf{r}$ implies $\overline{\Gamma(\rho_\mathbf{q}) \upharpoonright \mathbf{r}} \leqslant \Gamma(\rho_\mathbf{r}) \upharpoonright \mathbf{q}$. Hence by Definition 4.5.1 $(\Gamma; C)$ is consistent. □

Corollary A.1.0.1. *If $(\Gamma_1, \Gamma_2; C_1 \otimes C_2)$ is consistent, then $(\Gamma_1; C_1)$ and $(\Gamma_2; C_2)$ are consistent.*

Proof. By repeatedly applying Proposition A.1.5 and Proposition A.1.6 to remove all entries of $(\Gamma_1; C_1)$ from $(\Gamma_1, \Gamma_2; C_1 \otimes C_2)$, we prove that $(\Gamma_2; C_2)$ is consistent. By the symmetric procedure we prove that $(\Gamma_1; C_1)$ is consistent. □

Proposition A.1.7. *If $(\Gamma, \mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho), \rho : S;\ C)$ is consistent and $S \leqslant S'$, then $(\Gamma, \mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho), \rho : S';\ C)$ is consistent.*

Proof. Assume $(\Gamma, \mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho), \rho : S;\ C)$ is consistent, and take any S' such that $S \leqslant S'$. By Definition 4.5.1 we know that $\forall \mathtt{s}[\mathbf{q}] : \mathtt{tr}(\rho_\mathbf{q}), \rho_\mathbf{q} : \{\rho_\mathbf{q} \mapsto S_\mathbf{q}\} \in \mathrm{dom}\,(\Gamma, \mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho), \rho : \{\rho \mapsto S\})$: $\overline{S_\mathbf{q} \upharpoonright \mathbf{p}} \leqslant S \upharpoonright \mathbf{q}$; moreover, by Proposition A.1.2 we have $\forall \mathbf{q} : S \upharpoonright \mathbf{q} \leqslant S' \upharpoonright \mathbf{q}$. By transitivity of $\leqslant$ we also have $\forall \mathtt{s}[\mathbf{q}] : \mathtt{tr}(\rho_\mathbf{q}), \rho_\mathbf{q} : \{\rho_\mathbf{q} \mapsto S_\mathbf{q}\} \in \mathrm{dom}\,(\Gamma, \mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho), \rho : \{\rho \mapsto S\})$: $\overline{S_\mathbf{q} \upharpoonright \mathbf{p}} \leqslant S' \upharpoonright \mathbf{q}$. Therefore by Definition 4.5.1 we conclude that $(\Gamma, \mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho), \rho : S';\ C)$ is consistent. □

Corollary A.1.0.2. *If $(\Gamma_1, \Gamma_2;\ C_1 \otimes C_2)$ is consistent and $C_2 \leqslant C_2'$, then $(\Gamma_1, \Gamma_2';\ C_1 \otimes C_2')$ is consistent.*

Proof. By induction on the size of Γ_2. The base case: $\Gamma_2 = \varnothing$ is trivial, while the inductive case is proved by the induction hypothesis, and Proposition A.1.7. □

Lemma 4.5.1. *If $(\Gamma;\ C) \longrightarrow (\Gamma';\ C')$ and $(\Gamma;\ C)$ is* consistent (resp. complete)*, then so is $(\Gamma';\ C')$.*

Lemma 4.5.1. By induction on $(\Gamma;\ C) \longrightarrow (\Gamma'; C')$, as per Definition 4.5.2.

- **base case:**

$$(\Gamma;\ C) =$$
$$(\mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho_\mathbf{p}),\ \mathtt{s}[\mathbf{q}] : \mathtt{tr}(\rho_\mathbf{q}), \rho_\mathbf{p} : \{\rho_\mathbf{p} \mapsto S_\mathbf{p}\}, \rho_\mathbf{q} : \{\rho_\mathbf{q} \mapsto S_\mathbf{q}\};\ \{\rho_\mathbf{p} \mapsto S_\mathbf{p}, \rho_\mathbf{q} \mapsto S_\mathbf{q}\}) \longrightarrow$$
$$(\mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho_\mathbf{p}),\ \mathtt{s}[\mathbf{q}] : \mathtt{tr}(\rho_\mathbf{q}), \rho_\mathbf{p} : \{\rho_\mathbf{p} \mapsto S_k\}, \rho_\mathbf{q} : \{\rho_\mathbf{q} \mapsto S_k'\};\ \{\rho_\mathbf{p} \mapsto S_k, \rho_\mathbf{q} \mapsto S_k'\}) =$$
$$(\Gamma'; C')$$

with $\mathtt{unf}(S_\mathbf{p}) = \mathbf{q} \oplus_{i \in I} !l_i(U_i).S_i$, $\mathtt{unf}(S_\mathbf{q}) = \mathbf{p} \,\&_{i \in I \cup J}\, ?l_i(U_i').S_i'$, $k \in I$ and $U_k \leqslant U_k'$

We observe:

$$\overline{S_{\mathbf{p}} \upharpoonright \mathbf{q}} \leqslant S_{\mathbf{q}} \upharpoonright \mathbf{p} \quad \text{by the hypothesis and Definition 4.5.1} \quad (A.1)$$

$$\mathtt{unf}(\overline{S_{\mathbf{p}} \upharpoonright \mathbf{q}}) \leqslant \mathtt{unf}(S_{\mathbf{q}} \upharpoonright \mathbf{p}) \quad \text{by [SPARμL] and [SPARμR]} \quad (A.2)$$

$$\overline{\mathtt{unf}(S_{\mathbf{p}} \upharpoonright \mathbf{q})} \leqslant \mathtt{unf}(S_{\mathbf{q}} \upharpoonright \mathbf{p}) \quad \text{by Proposition A.1.1} \quad (A.3)$$

$$\overline{\mathtt{unf}(S_{\mathbf{p}}) \upharpoonright \mathbf{q}} \leqslant \mathtt{unf}(S_{\mathbf{q}}) \upharpoonright \mathbf{p} \quad \text{by Proposition A.1.3} \quad (A.4)$$

$$\overline{\mathbf{q} \oplus_{i\in I} !l_i(U_i).S_i \upharpoonright \mathbf{q}} \leqslant \mathbf{p} \,\&_{i\in I\cup J}\, ?l_i(U'_i).S'_i \upharpoonright \mathbf{p} \quad \text{by the hypothesis} \quad (A.5)$$

$$\overline{\oplus_{i\in I} !l_i(U_i).(S_i \upharpoonright \mathbf{q})} \leqslant \,\&_{i\in I\cup J}\, ?l_i(U'_i).S'_i \upharpoonright \mathbf{p} \quad \text{by Definition 4.3.4} \quad (A.6)$$

$$\&_{i\in I} ?l_i(U_i).\overline{(S_i \upharpoonright \mathbf{q})} \leqslant \,\&_{i\in I\cup J}\, ?l_i(U'_i).S'_i \upharpoonright \mathbf{p} \quad \text{by Definition 4.3.3} \quad (A.7)$$

$$\forall k \in I : \overline{S_k \upharpoonright \mathbf{q}} \leqslant S'_k \upharpoonright \mathbf{p} \quad \text{by [SPARBR]} \quad (A.8)$$

$$\text{if } \mathtt{unf}(S_{\mathbf{p}}) = \mathbf{q} \oplus_{i\in I} !l_i(\exists[\rho_i|\{\rho_i \mapsto U_i\}].\mathtt{tr}(\rho_i)).S_i, \text{ and}$$
$$\mathtt{unf}(S_{\mathbf{q}}) = \mathbf{p} \,\&_{i\in I\cup J}\, ?l_i(\exists[\rho_i|\{\rho_i \mapsto U'_i\}].\mathtt{tr}(\rho_i)).S'_i,\ k \in I \text{ with } U_k \leqslant U'_k$$

$$\overline{\mathbf{q} \oplus_{i\in I} !l_i(\exists[\rho_i|\{\rho_i \mapsto U_i\}].\mathtt{tr}(\rho_i)).S_i \upharpoonright \mathbf{q}} \leqslant \mathbf{p} \,\&_{i\in I\cup J}\, ?l_i(\exists[\rho_i|\{\rho_i \mapsto U'_i\}].\mathtt{tr}(\rho_i)).S'_i \upharpoonright \mathbf{p} \quad \text{by the hypothesis} \quad (A.9)$$

$$\overline{\oplus_{i\in I} !l_i(\exists[\rho_i|\{\rho_i \mapsto U_i\}].\mathtt{tr}(\rho_i)).(S_i \upharpoonright \mathbf{q})} \leqslant \,\&_{i\in I\cup J}\, ?l_i(\exists[\rho_i|\{\rho_i \mapsto U'_i\}].\mathtt{tr}(\rho_i)).S'_i \upharpoonright \mathbf{p} \quad \text{by Definition 4.3.4} \quad (A.10)$$

$$\&_{i\in I} ?l_i(\exists[\rho_i|\{\rho_i \mapsto U_i\}].\mathtt{tr}(\rho_i)).\overline{(S_i \upharpoonright \mathbf{q})} \leqslant \,\&_{i\in I\cup J}\, ?l_i(\exists[\rho_i|\{\rho_i \mapsto U'_i\}].\mathtt{tr}(\rho_i)).S'_i \upharpoonright \mathbf{p} \quad \text{by Definition 4.3.3} \quad (A.11)$$

$$\forall k \in I : \overline{S_k \upharpoonright \mathbf{q}} \leqslant S'_k \upharpoonright \mathbf{p} \quad \text{by [SPARBRP]} \quad (A.12)$$

and we conclude that $(\Gamma'; C')$ is consistent;

- **inductive case**:

$$(\Gamma;\, C) = (\Gamma_1, \mathtt{s}[\mathbf{r}] : \mathtt{tr}(\rho), \rho : \{\rho \mapsto S\};\, C_1 \otimes \{\rho \mapsto S\}) \longrightarrow$$
$$({\Gamma_1}', \mathtt{s}[\mathbf{r}] : \mathtt{tr}(\rho), \rho : \{\rho \mapsto S'\};\, {C_1}' \otimes \{\rho \mapsto S'\}) = (\Gamma';\, C') \text{ with } S \leqslant S' \quad (A.13)$$

We observe that $(\Gamma_1;\, C_1), (\Gamma'_1;\, C'_1)$ must have the form:

$$(\Gamma_1;\, C_1) = (\mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho_{\mathbf{p}}), \mathtt{s}[\mathbf{q}] : \mathtt{tr}(\rho_{\mathbf{q}}), \rho_{\mathbf{p}} : \{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}\}, \rho_{\mathbf{q}} : \{\rho_{\mathbf{q}} \mapsto S_{\mathbf{q}}\}, \Gamma_0;$$
$$\{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}, \rho_{\mathbf{q}} \mapsto S_{\mathbf{q}}\} \otimes C_0) \quad (A.14)$$

$$(\Gamma'_1; C'_1) = (\mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho_{\mathbf{p}}), \mathtt{s}[\mathbf{q}] : \mathtt{tr}(\rho_{\mathbf{q}}), \rho_{\mathbf{p}} : \{\rho_{\mathbf{p}} \mapsto S'_{\mathbf{p}}\}, \rho_{\mathbf{q}} : \{\rho_{\mathbf{q}} \mapsto S'_{\mathbf{q}}\}, \Gamma'_0;$$
$$\{\rho_{\mathbf{p}} \mapsto S'_{\mathbf{p}}, \rho_{\mathbf{q}} \mapsto S'_{\mathbf{q}}\} \otimes C'_0) \quad (A.15)$$

where

$$(\mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho_{\mathbf{p}}), \mathtt{s}[\mathbf{q}] : \mathtt{tr}(\rho_{\mathbf{q}}), \rho_{\mathbf{p}} : \{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}\}, \rho_{\mathbf{q}} : \{\rho_{\mathbf{q}} \mapsto S_{\mathbf{q}}\}; \{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}, \rho_{\mathbf{q}} \mapsto S_{\mathbf{q}}\}) \longrightarrow$$
$$(\mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho_{\mathbf{p}}), \mathtt{s}[\mathbf{q}] : \mathtt{tr}(\rho_{\mathbf{q}}), \rho_{\mathbf{p}} : \{\rho_{\mathbf{p}} \mapsto S'_{\mathbf{p}}\}, \rho_{\mathbf{q}} : \{\rho_{\mathbf{q}} \mapsto S'_{\mathbf{q}}\}; \{\rho_{\mathbf{p}} \mapsto S'_{\mathbf{p}}, \rho_{\mathbf{q}} \mapsto S'_{\mathbf{q}}\}) \quad \text{(A.16)}$$

and $C_0 \leqslant C'_0$.

Therefore:

$$(\Gamma;\, C) = (\Gamma_1, \mathtt{s}[\mathbf{r}] : \mathtt{tr}(\rho), \rho : \{\rho \mapsto S\};\, C_1 \otimes \{\rho \mapsto S\}) =$$
$$(\mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho_{\mathbf{p}}), \mathtt{s}[\mathbf{q}] : \mathtt{tr}(\rho_{\mathbf{q}}), \rho_{\mathbf{p}} : \{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}\}, \rho_{\mathbf{q}} : \{\rho_{\mathbf{q}} \mapsto S_{\mathbf{q}}\}, \Gamma_0, \mathtt{s}[\mathbf{r}] : \mathtt{tr}(\rho), \rho : \{\rho \mapsto S\};$$
$$\{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}, \rho_{\mathbf{q}} \mapsto S_{\mathbf{q}}\} \otimes C_0 \otimes \{\rho \mapsto S\}) \text{ is consistent; by the hypothesis} \quad \text{(A.17)}$$

$$(\mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho_{\mathbf{p}}), \mathtt{s}[\mathbf{q}] : \mathtt{tr}(\rho_{\mathbf{q}}), \rho_{\mathbf{p}} : \{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}\}, \rho_{\mathbf{q}} : \{\rho_{\mathbf{q}} \mapsto S_{\mathbf{q}}\}, \Gamma_0, \mathtt{s}[\mathbf{r}] : \mathtt{tr}(\rho), \rho : \{\rho \mapsto S\};$$
$$\{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}, \rho_{\mathbf{q}} \mapsto S_{\mathbf{q}}\} \otimes C'_0 \otimes \{\rho \mapsto S'\})$$
$$\text{is consistent; from the above and Corollary A.1.0.2} \quad \text{(A.18)}$$

$$(\Gamma_0, \mathtt{s}[\mathbf{r}] : \mathtt{tr}(\rho), \rho : \{\rho \mapsto S\}; C'_0 \otimes \{\rho \mapsto S'\})$$
$$\text{is consistent; from the above and Corollary A.1.0.1} \quad \text{(A.19)}$$

□

Corollary 4.5.1. *If* $(\Gamma_1, \Gamma_2; C_1 \otimes C_2)$ *is consistent and* $(\Gamma_1; C_1) \longrightarrow^* (\Gamma'_1; C'_1)$*, then* $(\Gamma'_1, \Gamma_2; C'_1 \otimes C_2)$ *is consistent.*

Proof of Corollary 4.5.1. Assume all the hypotheses, and let n be the length of the sequence of reductions in $(\Gamma_1;\, C_1) \longrightarrow^* (\Gamma'_1; C'_1)$. In the base case, $n = 0$, the thesis holds trivially. In the inductive case $n = n' + 1$, we have:

$$(\Gamma_1, C_1) \longrightarrow \ldots \longrightarrow (\Gamma^*_1, C^*_1) \longrightarrow (\Gamma'_1, C'_1)$$

and by the induction hypothesis,$(\Gamma^*_1, \Gamma_2;\, C^*_1 \otimes C_2)$ is consistent. This implies that $(\Gamma'_1, \Gamma_2; C'_1 \otimes C_2)$ is consistent: we prove such a fact with a further induction on the size of $(\Gamma_2;\, C_2)$. In the base case $(\Gamma_2;\, C_2) = (\varnothing; \varnothing)$ we conclude immediately by Lemma 4.5.1. In the inductive case we have $(\Gamma_2;\, C_2) = (\Gamma_0, \mathtt{c} : \mathtt{tr}(\rho), \rho : \{\rho \mapsto U\};\, C_0 \otimes \{\rho \mapsto U\})$; by applying the induction hypothesis we get that $(\Gamma'_1, \Gamma_0; C'_1 \otimes C_0)$ is consistent. We examine the shape of the additional entry $(\mathtt{c} : \mathtt{tr}(\rho), \rho : \{\rho \mapsto U\}; \{\rho \mapsto U\})$ and its consistency w.r.t. $(\Gamma'_1, \Gamma_0; C'_1 \otimes C_0)$, similarly to the inductive case in the proof of Lemma 4.5.1. In all cases, we conclude that $(\Gamma'_1, \Gamma_2; C'_1 \otimes C_2)$ is consistent. □

Proposition 4.5.1 (Weakening)**.** *For any multiparty session process* P *with* $\Delta; \Gamma \vdash P;\, C$*:*

1. *if* Δ *and* Δ' *are disjoint, then* $\Delta, \Delta'; \Gamma \vdash P;\, C$.

2. *if* Γ *and* Γ' *are disjoint, then* $\Delta;\Gamma,\Gamma' \vdash P;\ C$.

Proof of Proposition 4.5.1. By induction on typing derivations, with a case analysis on the last rule applied.

- case $P = \mathbf{0}$ By applying [TINACT] on $\Delta,\Delta';\Gamma \vdash P;\ C$, respectively $\Delta;\Gamma,\Gamma' \vdash P;\ C$.

- case $P = P \mid Q$

  1. $\Delta,\Delta';\Gamma \vdash P \mid Q;\ C$.

     Let $C = C_1 \otimes C_2$. From the premise we know that $\Delta;\Gamma \vdash P \mid Q;\ C_1 \otimes C_2$.

     $$\frac{\Delta;\Gamma \vdash P;C_1 \quad \Delta;\Gamma \vdash Q;C_2}{\Delta;\Gamma \vdash P \mid Q;C_1 \otimes C_2}\ [\text{TPAR}]$$

     By applying [TPAR] we obtain $\Delta;\Gamma \vdash P;C_1$ and $\Delta;\Gamma \vdash Q;C_2$. By applying the inductive hypothesis on these two we obtain $\Delta,\Delta';\Gamma \vdash P;C_1$, respectively $\Delta,\Delta';\Gamma \vdash Q;C_2$. By applying [TPAR] we obtain our conclusion:

     $$\frac{\Delta,\Delta';\Gamma \vdash P;C_1 \quad \Delta,\Delta';\Gamma \vdash Q;C_2}{\Delta,\Delta';\Gamma \vdash P \mid Q;C_1 \otimes C_2}\ [\text{TPAR}]$$

  2. $\Delta;\Gamma,\Gamma' \vdash P \mid Q;\ C$.

     Let $C = C_1 \otimes C_2$. From the premise we know that $\Delta;\Gamma \vdash P \mid Q;\ C_1 \otimes C_2$.

     $$\frac{\Delta;\Gamma \vdash P;C_1 \quad \Delta;\Gamma \vdash Q;C_2}{\Delta;\Gamma \vdash P \mid Q;C_1 \otimes C_2}\ [\text{TPAR}]$$

     By applying [TPAR] we obtain $\Delta;\Gamma \vdash P;C_1$ and $\Delta;\Gamma \vdash Q;C_2$. By applying the inductive hypothesis on these two we obtain $\Delta;\Gamma,\Gamma' \vdash P;C_1$, respectively $\Delta;\Gamma,\Gamma' \vdash Q;C_2$. By applying [TPAR] we obtain our conclusion:

     $$\frac{\Delta;\Gamma,\Gamma' \vdash P;C_1 \quad \Delta;\Gamma,\Gamma' \vdash Q;C_2}{\Delta;\Gamma,\Gamma' \vdash P \mid Q;C_1 \otimes C_2}\ [\text{TPAR}]$$

- case $P = (\nu\, s)P$

  1. $\Delta,\Delta';\Gamma \vdash (\nu\, s)P;\ C$

     From the premise we know that $\Delta;\Gamma \vdash (\nu\, s)P;C$.

     $$\frac{\Delta;\Gamma,\Gamma' \vdash P;C \otimes C' \qquad (\Gamma' = \{\mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho_{\mathbf{p}}), \rho_{\mathbf{p}} : \{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}\}\}_{\mathbf{p}\in I}, C' = \otimes_{\mathbf{p}\in I}\{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}\})\ \text{complete}}{\Delta;\Gamma \vdash (\nu\, s : \Gamma')P;C}\ [\text{TRES}]$$

     By applying [TRES] we obtain $\Delta;\Gamma,\Gamma' \vdash P;C \otimes C'$ with $(\Gamma' = \{\mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho_{\mathbf{p}}), \rho_{\mathbf{p}} : \{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}\}\}_{\mathbf{p}\in I}, C' = \otimes_{\mathbf{p}\in I}\{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}\})$ complete. By applying the induction hypothesis we obtain: $\Delta,\Delta';\Gamma,\Gamma' \vdash P;C \otimes C'$. By applying [TINACT] we obtain our conclusion:

$$\frac{\Delta,\Delta';\Gamma,\Gamma' \vdash P;C\otimes C' \quad (\Gamma' = \{\mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho_{\mathbf{p}}), \rho_{\mathbf{p}} : \{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}\}\}_{\mathbf{p}\in I}, C' = \otimes_{\mathbf{p}\in I}\{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}\}) \text{ complete}}{\Delta,\Delta';\Gamma \vdash (\nu\, s : \Gamma')P;C}\ [\text{TRES}]$$

2. $\Delta;\Gamma,\Gamma_1 \vdash (\nu\, s)P;\ C$

   From the premise we know that $\Delta;\Gamma \vdash (\nu\, s)P;C$.

$$\frac{\Delta;\Gamma,\Gamma' \vdash P;C\otimes C' \quad (\Gamma' = \{\mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho_{\mathbf{p}}), \rho_{\mathbf{p}} : \{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}\}\}_{\mathbf{p}\in I}, C' = \otimes_{\mathbf{p}\in I}\{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}\}) \text{ complete}}{\Delta;\Gamma \vdash (\nu\, s : \Gamma')P;C}\ [\text{TRES}]$$

   By applying [TINACT] we obtain $\Delta;\Gamma,\Gamma' \vdash P;C\otimes C'$ with $(\Gamma' = \{\mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho_{\mathbf{p}}), \rho_{\mathbf{p}} : \{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}\}\}_{\mathbf{p}\in I}, C' = \otimes_{\mathbf{p}\in I}\{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}\})$ complete. By applying the induction hypothesis we obtain: $\Delta;\Gamma,\Gamma',\Gamma_1 \vdash P;C\otimes C'$. By applying [TINACT] we obtain our conclusion:

$$\frac{\Delta;\Gamma,\Gamma',\Gamma_1 \vdash P;C\otimes C' \quad (\Gamma' = \{\mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho_{\mathbf{p}}), \rho_{\mathbf{p}} : \{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}\}\}_{\mathbf{p}\in I}, C' = \otimes_{\mathbf{p}\in I}\{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}\}) \text{ complete}}{\Delta;\Gamma,\Gamma_1 \vdash (\nu\, s : \Gamma')P;C}\ [\text{TRES}]$$

- case $P = \mathtt{c}[\mathbf{p}] \oplus \langle l(v)\rangle.P$

1. $\Delta,\Delta';\Gamma \vdash \mathtt{c}[\mathbf{p}] \oplus \langle l(v)\rangle.P;\ C$

   From the premise we know that $\Delta;\Gamma \vdash \mathtt{c}[\mathbf{p}] \oplus \langle l(v)\rangle.P;C$.

$$\frac{\Gamma \vdash v : U;C \quad \Delta;\Gamma \vdash P;C'\otimes\{\rho \mapsto S_j\} \quad \mathtt{c} : \mathtt{tr}(\rho), \rho : \{\rho \mapsto S_j\} \in \Gamma \quad j \in I}{\Delta;\Gamma \vdash \mathtt{c}[\mathbf{p}] \oplus \langle l_j(v)\rangle.P;C\otimes C'\otimes\{\rho \mapsto \mathbf{p} \oplus_{i\in I} !l_i(U_i).S_i\}}\ [\text{TSEL}]$$

   By applying [TSEL] we obtain $\Delta;\Gamma \vdash P;C'\otimes\{\rho \mapsto S_j\}$. By applying the induction hypothesis we obtain: $\Delta,\Delta';\Gamma \vdash P;C'\otimes\{\rho \mapsto S_j\}$. By applying [TSEL] we obtain our conclusion:

$$\frac{\Gamma \vdash v : U;C \quad \Delta,\Delta';\Gamma \vdash P;C'\otimes\{\rho \mapsto S_j\} \quad \mathtt{c} : \mathtt{tr}(\rho), \rho : \{\rho \mapsto S_j\} \in \Gamma \quad j \in I}{\Delta,\Delta';\Gamma \vdash \mathtt{c}[\mathbf{p}] \oplus \langle l_j(v)\rangle.P;C\otimes C'\otimes\{\rho \mapsto \mathbf{p} \oplus_{i\in I} !l_i(U_i).S_i\}}\ [\text{TSEL}]$$

2. $\Delta;\Gamma,\Gamma' \vdash \mathtt{c}[\mathbf{p}] \oplus \langle l(v)\rangle.P;\ C$

   From the premise we know that $\Delta;\Gamma \vdash \mathtt{c}[\mathbf{p}] \oplus \langle l(v)\rangle.P;C$.

$$\frac{\Gamma \vdash v : U;C \quad \Delta;\Gamma \vdash P;C'\otimes\{\rho \mapsto S_j\} \quad \mathtt{c} : \mathtt{tr}(\rho), \rho : \{\rho \mapsto S_j\} \in \Gamma \quad j \in I}{\Delta;\Gamma \vdash \mathtt{c}[\mathbf{p}] \oplus \langle l_j(v)\rangle.P;C\otimes C'\otimes\{\rho \mapsto \mathbf{p} \oplus_{i\in I} !l_i(U_i).S_i\}}\ [\text{TSEL}]$$

   By applying [TSEL] we obtain $\Gamma \vdash v : U;C$ and $\Delta;\Gamma \vdash P;C'\otimes\{\rho \mapsto S_j\}$. By applying the induction hypothesis we obtain $\Gamma,\Gamma' \vdash v : U;C$ and $\Delta;\Gamma,\Gamma' \vdash P;C'\otimes\{\rho \mapsto S_j\}$. By applying [TSEL] we obtain our conclusion:

$$\frac{\Gamma,\Gamma' \vdash v : U;C \quad \Delta;\Gamma,\Gamma' \vdash P;C'\otimes\{\rho \mapsto S_j\} \quad \mathtt{c} : \mathtt{tr}(\rho), \rho : \{\rho \mapsto S_j\} \in \Gamma \quad j \in I}{\Delta;\Gamma,\Gamma' \vdash \mathtt{c}[\mathbf{p}] \oplus \langle l_j(v)\rangle.P;C\otimes C'\otimes\{\rho \mapsto \mathbf{p} \oplus_{i\in I} !l_i(U_i).S_i\}}\ [\text{TSEL}]$$

- case $P = \mathtt{c}[\mathbf{p}]\&_{i\in I}\{l_i(x_i).P_i\}$

1. $\Delta,\Delta';\Gamma\vdash \mathsf{c}[\mathbf{p}]\&_{i\in I}\{l_i(x_i).P_i\};\ C$

   From the premise we know that $\Delta;\Gamma\vdash \mathsf{c}[\mathbf{p}]\&_{i\in I}\{l_i(x_i).P_i\};C$.

   $$\frac{\Delta;\Gamma,x_i:U_i\vdash P_i;C\otimes C_i\otimes\{\rho\mapsto S_i\}\quad \mathsf{c}:\mathtt{tr}(\rho),\rho:\{\rho\mapsto S_i\}\in\Gamma\quad\forall i\in I}{\Delta;\Gamma\vdash \mathsf{c}[\mathbf{p}]\&_{i\in I}\{l_i(x_i).P_i\};C\otimes\{\rho\mapsto \mathbf{p}\ \&_{i\in I}?l_i(U_i).S_i\}}\ [\text{TBR}]$$

   By applying [TBR] we obtain $\Delta;\Gamma,x_i:U_i\vdash P_i;C\otimes C_i\otimes\{\rho\mapsto S_i\}$. By applying the induction hypothesis we obtain $\Delta,\Delta';\Gamma,x_i:U_i\vdash P_i;C\otimes C_i\otimes\{\rho\mapsto S_i\}$. By applying [TBR] we obtain:

   $$\frac{\Delta,\Delta';\Gamma,x_i:U_i\vdash P_i;C\otimes C_i\otimes\{\rho\mapsto S_i\}\quad \mathsf{c}:\mathtt{tr}(\rho),\rho:\{\rho\mapsto S_i\}\in\Gamma\quad\forall i\in I}{\Delta,\Delta';\Gamma\vdash \mathsf{c}[\mathbf{p}]\&_{i\in I}\{l_i(x_i).P_i\};C\otimes\{\rho\mapsto \mathbf{p}\ \&_{i\in I}?l_i(U_i).S_i\}}\ [\text{TBR}]$$

2. $\Delta;\Gamma,\Gamma'\vdash \mathsf{c}[\mathbf{p}]\&_{i\in I}\{l_i(x_i).P_i\};\ C$

   From the premise we know that $\Delta;\Gamma\vdash \mathsf{c}[\mathbf{p}]\&_{i\in I}\{l_i(x_i).P_i\};C$.

   $$\frac{\Delta;\Gamma,x_i:U_i\vdash P_i;C\otimes C_i\otimes\{\rho\mapsto S_i\}\quad \mathsf{c}:\mathtt{tr}(\rho),\rho:\{\rho\mapsto S_i\}\in\Gamma\quad\forall i\in I}{\Delta;\Gamma\vdash \mathsf{c}[\mathbf{p}]\&_{i\in I}\{l_i(x_i).P_i\};C\otimes\{\rho\mapsto \mathbf{p}\ \&_{i\in I}?l_i(U_i).S_i\}}\ [\text{TBR}]$$

   By applying [TBR] we obtain $\Delta;\Gamma,x_i:U_i\vdash P_i;C\otimes C_i\otimes\{\rho\mapsto S_i\}$. By applying the induction hypothesis we obtain $\Delta;\Gamma,x_i:U_i,\Gamma'\vdash P_i;C\otimes C_i\otimes\{\rho\mapsto S_i\}$. By applying [TBR] we obtain:

   $$\frac{\Delta;\Gamma,x_i:U_i,\Gamma'\vdash P_i;C\otimes C_i\otimes\{\rho\mapsto S_i\}\quad \mathsf{c}:\mathtt{tr}(\rho),\rho:\{\rho\mapsto S_i\}\in\Gamma,\Gamma'\quad\forall i\in I}{\Delta;\Gamma,\Gamma'\vdash \mathsf{c}[\mathbf{p}]\&_{i\in I}\{l_i(x_i).P_i\};C\otimes\{\rho\mapsto \mathbf{p}\ \&_{i\in I}?l_i(U_i).S_i\}}\ [\text{TBR}]$$

- case $P=\mathsf{c}[\mathbf{p}]\oplus\langle l(\mathsf{pack}(\rho,\mathsf{s}[\mathbf{q}]))\rangle.P$

1. $\Delta,\Delta';\Gamma\vdash \mathsf{c}[\mathbf{p}]\oplus\langle l(\mathsf{pack}(\rho,\mathsf{s}[\mathbf{q}]))\rangle.P;\ C$

   From the premise we know that $\Delta;\Gamma\vdash \mathsf{c}[\mathbf{p}]\oplus\langle l(\mathsf{pack}(\rho,\mathsf{s}[\mathbf{q}]))\rangle.P;C$.

   [TSELP]
   $$\frac{\Gamma\vdash v:\mathtt{tr}(\rho');\varnothing\quad \Delta;\Gamma\vdash P;C\otimes\{\rho\mapsto S_j,\rho'\mapsto U\}\quad \mathsf{c}:\mathtt{tr}(\rho),\rho:\{\rho\mapsto S_j\}\in\Gamma\quad j\in I}{\Delta;\Gamma\vdash \mathsf{c}[\mathbf{p}]\oplus\langle l_j(\mathsf{pack}(\rho',v))\rangle.P;C\otimes\{\rho\mapsto \mathbf{p}\oplus_{i\in I}!l_i(\exists[\rho'|\{\rho'\mapsto U\}].\mathtt{tr}(\rho')).S_i,\rho'\mapsto U\}}$$

   By applying [TSELP] we obtain $\Delta;\Gamma\vdash P;C\otimes\{\rho\mapsto S_j,\rho'\mapsto U\}$. By applying the induction hypothesis we obtain $\Delta,\Delta';\Gamma\vdash P;C\otimes\{\rho\mapsto S_j,\rho'\mapsto U\}$. By applying [TSELP] we obtain our conclusion:
   [TSELP]
   $$\frac{\Gamma\vdash v:\mathtt{tr}(\rho');\varnothing\quad \Delta,\Delta';\Gamma\vdash P;C\otimes\{\rho\mapsto S_j,\rho'\mapsto U\}\quad \mathsf{c}:\mathtt{tr}(\rho),\rho:\{\rho\mapsto S_j\}\in\Gamma\quad j\in I}{\Delta,\Delta';\Gamma\vdash \mathsf{c}[\mathbf{p}]\oplus\langle l_j(\mathsf{pack}(\rho',v))\rangle.P;C\otimes\{\rho\mapsto \mathbf{p}\oplus_{i\in I}!l_i(\exists[\rho'|\{\rho'\mapsto U\}].\mathtt{tr}(\rho')).S_i,\rho'\mapsto U\}}$$

2. $\Delta;\Gamma,\Gamma'\vdash \mathsf{c}[\mathbf{p}]\oplus\langle l(\mathsf{pack}(\rho,\mathsf{s}[\mathbf{q}]))\rangle.P;\ C$

   From the premise we know that $\Delta;\Gamma\vdash \mathsf{c}[\mathbf{p}]\oplus\langle l(\mathsf{pack}(\rho,\mathsf{s}[\mathbf{q}]))\rangle.P;C$.

   [TSELP]
   $$\frac{\Gamma\vdash v:\mathtt{tr}(\rho');\varnothing\quad \Delta;\Gamma\vdash P;C\otimes\{\rho\mapsto S_j,\rho'\mapsto U\}\quad \mathsf{c}:\mathtt{tr}(\rho),\rho:\{\rho\mapsto S_j\}\in\Gamma\quad j\in I}{\Delta;\Gamma\vdash \mathsf{c}[\mathbf{p}]\oplus\langle l_j(\mathsf{pack}(\rho',v))\rangle.P;C\otimes\{\rho\mapsto \mathbf{p}\oplus_{i\in I}!l_i(\exists[\rho'|\{\rho'\mapsto U\}].\mathtt{tr}(\rho')).S_i,\rho'\mapsto U\}}$$

By applying [TSELP] we obtain $\Delta;\Gamma \vdash P;C\otimes\{\rho \mapsto S_j, \rho' \mapsto U\}$ and $\Gamma \vdash v : \mathtt{tr}(\rho');\varnothing$. By applying the induction hypothesis we obtain $\Delta;\Gamma,\Gamma' \vdash P;C\otimes\{\rho \mapsto S_j, \rho' \mapsto U\}$ and $\Gamma,\Gamma' \vdash v : \mathtt{tr}(\rho');\varnothing$. By applying [TSELP] we obtain our conclusion:

[TSELP]

$$\frac{\Gamma,\Gamma' \vdash v : \mathtt{tr}(\rho');\varnothing \quad \Delta;\Gamma,\Gamma' \vdash P;C\otimes\{\rho \mapsto S_j, \rho' \mapsto U\} \quad \mathsf{c} : \mathtt{tr}(\rho), \rho : \{\rho \mapsto S_j\} \in \Gamma,\Gamma' \quad j \in I}{\Delta;\Gamma,\Gamma' \vdash \mathsf{c}[\mathbf{p}] \oplus \langle l_j(\mathsf{pack}(\rho', v))\rangle.P;C\otimes\{\rho \mapsto \mathbf{p} \oplus_{i\in I} !l_i(\exists[\rho'|\{\rho' \mapsto U\}].\mathtt{tr}(\rho')).S_i, \rho' \mapsto U\}}$$

- case $P = \mathsf{c}[\mathbf{p}]\&_{i\in I}\{l_i(\mathsf{pack}(\rho_i, \mathtt{s}_i[\mathbf{q}])).P_i\}$

  1. $\Delta,\Delta';\Gamma \vdash \mathsf{c}[\mathbf{p}]\&_{i\in I}\{l_i(\mathsf{pack}(\rho_i, \mathtt{s}_i[\mathbf{q}])).P_i\};\ C$

     From the premise we know that $\Delta;\Gamma \vdash \mathsf{c}[\mathbf{p}]\&_{i\in I}\{l_i(\mathsf{pack}(\rho_i, \mathtt{s}_i[\mathbf{q}])).P_i\};\ C$.

     [TBRP]

     $$\frac{\Delta;\Gamma, \mathtt{v}_i : \mathtt{tr}(\rho_i), \rho_i : \{\rho_i \mapsto U_i\} \vdash P_i;C\otimes\{\rho \mapsto S_i\} \quad \forall i \in I \quad \mathsf{c} : \mathtt{tr}(\rho), \rho : \{\rho \mapsto S_i\} \in \Gamma}{\Delta;\Gamma \vdash \mathsf{c}[\mathbf{p}]\&_{i\in I}\{l_i(\mathsf{pack}(\rho_i, \mathtt{v}_i)).P_i\};C\otimes\{\rho \mapsto \mathbf{p} \&_{i\in I} ?l_i(\exists[\rho_i|\{\rho_i \mapsto U_i\}].\mathtt{tr}(\rho_i)).S_i\}}$$

     By applying [TBRP] we obtain $\Delta;\Gamma, \mathtt{v}_i : \mathtt{tr}(\rho_i), \rho_i : \{\rho_i \mapsto U_i\} \vdash P_i;C\otimes\{\rho \mapsto S_i\}$. By applying the induction hypothesis we obtain $\Delta,\Delta';\Gamma, \mathtt{v}_i : \mathtt{tr}(\rho_i), \rho_i : \{\rho_i \mapsto U_i\} \vdash P_i;C\otimes\{\rho \mapsto S_i\}$. By applying [TBRP] we obtain our conclusion:

     [TBRP]

     $$\frac{\Delta,\Delta';\Gamma, \mathtt{v}_i : \mathtt{tr}(\rho_i), \rho_i : \{\rho_i \mapsto U_i\} \vdash P_i;C\otimes\{\rho \mapsto S_i\} \quad \forall i \in I \quad \mathsf{c} : \mathtt{tr}(\rho), \rho : \{\rho \mapsto S_i\} \in \Gamma}{\Delta,\Delta';\Gamma \vdash \mathsf{c}[\mathbf{p}]\&_{i\in I}\{l_i(\mathsf{pack}(\rho_i, \mathtt{v}_i)).P_i\};C\otimes\{\rho \mapsto \mathbf{p} \&_{i\in I} ?l_i(\exists[\rho_i|\{\rho_i \mapsto U_i\}].\mathtt{tr}(\rho_i)).S_i\}}$$

  2. $\Delta;\Gamma,\Gamma' \vdash \mathsf{c}[\mathbf{p}]\&_{i\in I}\{l_i(\mathsf{pack}(\rho_i, \mathtt{s}_i[\mathbf{q}])).P_i\};\ C$

     From the premise we know that $\Delta;\Gamma \vdash \mathsf{c}[\mathbf{p}]\&_{i\in I}\{l_i(\mathsf{pack}(\rho_i, \mathtt{s}_i[\mathbf{q}])).P_i\};\ C$.

     [TBRP]

     $$\frac{\Delta;\Gamma, \mathtt{v}_i : \mathtt{tr}(\rho_i), \rho_i : \{\rho_i \mapsto U_i\} \vdash P_i;C\otimes\{\rho \mapsto S_i\} \quad \forall i \in I \quad \mathsf{c} : \mathtt{tr}(\rho), \rho : \{\rho \mapsto S_i\} \in \Gamma}{\Delta;\Gamma \vdash \mathsf{c}[\mathbf{p}]\&_{i\in I}\{l_i(\mathsf{pack}(\rho_i, \mathtt{v}_i)).P_i\};C\otimes\{\rho \mapsto \mathbf{p} \&_{i\in I} ?l_i(\exists[\rho_i|\{\rho_i \mapsto U_i\}].\mathtt{tr}(\rho_i)).S_i\}}$$

     By applying [TBRP] we obtain $\Delta;\Gamma, \mathtt{v}_i : \mathtt{tr}(\rho_i), \rho_i : \{\rho_i \mapsto U_i\} \vdash P_i;C\otimes\{\rho \mapsto S_i\}$. By applying the induction hypothesis we obtain $\Delta;\Gamma,\Gamma', \mathtt{v}_i : \mathtt{tr}(\rho_i), \rho_i : \{\rho_i \mapsto U_i\} \vdash P_i;C\otimes\{\rho \mapsto S_i\}$. By applying [TBRP] we obtain our conclusion:

     [TBRP]

     $$\frac{\Delta;\Gamma,\Gamma', \mathtt{v}_i : \mathtt{tr}(\rho_i), \rho_i : \{\rho_i \mapsto U_i\} \vdash P_i;C\otimes\{\rho \mapsto S_i\} \quad \forall i \in I \quad \mathsf{c} : \mathtt{tr}(\rho), \rho : \{\rho \mapsto S_i\} \in \Gamma,\Gamma'}{\Delta;\Gamma,\Gamma' \vdash \mathsf{c}[\mathbf{p}]\&_{i\in I}\{l_i(\mathsf{pack}(\rho_i, \mathtt{v}_i)).P_i\};C\otimes\{\rho \mapsto \mathbf{p} \&_{i\in I} ?l_i(\exists[\rho_i|\{\rho_i \mapsto U_i\}].\mathtt{tr}(\rho_i)).S_i\}}$$

- **def** D **in** P

  1. $\Delta,\Delta';\Gamma \vdash$ **def** D **in** $P;\ C$

     From the premise we know that $\Delta;\Gamma \vdash$ **def** D **in** $P;\ C$.

     $$\frac{\Delta, X : \tilde{U};\tilde{x} : \tilde{U} \vdash P;\tilde{C} \qquad \Delta, X : \tilde{U};\Gamma \vdash Q;C}{\Delta;\Gamma \vdash \mathbf{def}\ X\langle\tilde{x} : \tilde{U}\rangle = P;\tilde{C}\ \mathbf{in}\ Q;C}\ [\text{TDEF}]$$

     By applying [TDEF] we obtain $\Delta, X : \tilde{U};\tilde{x} : \tilde{U} \vdash P;\tilde{C}$ and $\Delta, X : \tilde{U};\Gamma \vdash Q;C$. By applying the induction hypothesis we obtain $\Delta, X : \tilde{U},\Delta';\tilde{x} : \tilde{U} \vdash P;\tilde{C}$ and $\Delta, X : \tilde{U},\Delta';\Gamma \vdash Q;C$. By applying [TDEF] we obtain our conclusion:

$$\frac{\Delta, X:\tilde{U}, \Delta'; \tilde{x}:\tilde{U} \vdash P; \tilde{C} \qquad \Delta, X:\tilde{U}, \Delta'; \Gamma \vdash Q; C}{\Delta, \Delta'; \Gamma \vdash \mathbf{def}\, X\langle \tilde{x}:\tilde{U}\rangle = P; \tilde{C}\ \mathbf{in}\ Q; C}\ [\text{TDEF}]$$

2. $\Delta; \Gamma, \Gamma' \vdash \mathbf{def}\, D\ \mathbf{in}\ P;\ C$

   From the premise we know that $\Delta; \Gamma \vdash \mathbf{def}\, D\ \mathbf{in}\ P;\ C$.

   $$\frac{\Delta, X:\tilde{U}; \tilde{x}:\tilde{U} \vdash P; \tilde{C} \qquad \Delta, X:\tilde{U}; \Gamma \vdash Q; C}{\Delta; \Gamma \vdash \mathbf{def}\, X\langle \tilde{x}:\tilde{U}\rangle = P; \tilde{C}\ \mathbf{in}\ Q; C}\ [\text{TDEF}]$$

   By applying [TDEF] we obtain $\Delta, X:\tilde{U}; \Gamma \vdash Q; C$. By applying the induction hypothesis we obtain $\Delta, X:\tilde{U}; \Gamma, \Gamma' \vdash Q; C$. By applying [TDEF] we obtain our conclusion:

   $$\frac{\Delta, X:\tilde{U}; \tilde{x}:\tilde{U} \vdash P; \tilde{C} \qquad \Delta, X:\tilde{U}; \Gamma, \Gamma' \vdash Q; C}{\Delta, \Delta'; \Gamma, \Gamma' \vdash \mathbf{def}\, X\langle \tilde{x}:\tilde{U}\rangle = P; \tilde{C}\ \mathbf{in}\ Q; C}\ [\text{TDEF}]$$

- $P = X\langle \tilde{x} \rangle$

1. $\Delta, \Delta'; \Gamma \vdash X\langle \tilde{x} \rangle;\ C$

   By applying [TCALL] we obtain our conclusion:

   [TCALL]
   $$\frac{\forall i \in \{1..n\} \quad \Gamma \vdash v_i : U_i; C_i}{\Delta, X : U_1, \ldots, U_n, \Delta'; \Gamma \vdash X\langle v_1, \ldots, v_n \rangle; C_1 \otimes \ldots \otimes C_n}$$

2. $\Delta; \Gamma, \Gamma' \vdash X\langle \tilde{x} \rangle;\ C$

   From the premise we know that $\Delta; \Gamma \vdash X\langle \tilde{x} \rangle;\ C$.

   [TCALL]
   $$\frac{\forall i \in \{1..n\} \quad \Gamma \vdash v_i : U_i; C_i}{\Delta, X : U_1, \ldots, U_n; \Gamma \vdash X\langle v_1, \ldots, v_n \rangle; C_1 \otimes \ldots \otimes C_n}$$

   By applying [TCALL] we obtain $\Gamma \vdash v_i : U_i; C_i$. By applying the induction hypothesis we obtain $\Gamma, \Gamma' \vdash v_i : U_i; C_i$. By applying [TCALL] we obtain our conclusion:

   [TCALL]
   $$\frac{\forall i \in \{1..n\} \quad \Gamma, \Gamma' \vdash v_i : U_i; C_i}{\Delta, X : U_1, \ldots, U_n; \Gamma, \Gamma' \vdash X\langle v_1, \ldots, v_n \rangle; C_1 \otimes \ldots \otimes C_n}$$

□

Proposition 4.5.2 (Strengthening)**.** *For any multiparty session process P:*

1. *if* $\Delta, \Delta'; \Gamma \vdash P;\ C$ *and* $\Delta' \notin fpv(P)$ *then* $\Delta; \Gamma \vdash P;\ C$.
2. *if* $\Delta; \Gamma, \Gamma' \vdash P;\ C$ *and* $\Gamma' \notin fv(P)$ *and* $\Gamma' \notin C$*, then* $\Delta; \Gamma \vdash P;\ C$.

Proof of Proposition 4.5.2. By induction on typing derivations, with a case analysis on the last rule applied.

- case $P = \mathbf{0}$ By applying [TINACT] on $\Delta; \Gamma \vdash P;\ C$.
- case $P = P \mid Q$

1. $\Delta,\Delta';\Gamma\vdash P\,|\,Q;\ C$.

   Let $C = C_1 \otimes C_2$.

   $$\frac{\Delta,\Delta';\Gamma\vdash P;C_1 \quad \Delta,\Delta';\Gamma\vdash Q;C_2}{\Delta,\Delta';\Gamma\vdash P\,|\,Q;C_1\otimes C_2}\ [\text{TPAR}]$$

   By applying [TPAR] we obtain $\Delta,\Delta';\Gamma\vdash P;C_1$, respectively $\Delta,\Delta';\Gamma\vdash Q;C_2$. By applying the inductive hypothesis on these two we obtain $\Delta;\Gamma\vdash P;C_1$ and $\Delta;\Gamma\vdash Q;C_2$. By applying [TPAR] we obtain our conclusion:

   $$\frac{\Delta;\Gamma\vdash P;C_1 \quad \Delta;\Gamma\vdash Q;C_2}{\Delta;\Gamma\vdash P\,|\,Q;C_1\otimes C_2}\ [\text{TPAR}]$$

2. $\Delta;\Gamma,\Gamma'\vdash P\,|\,Q;\ C$.

   $$\frac{\Delta;\Gamma,\Gamma'\vdash P;C_1 \quad \Delta;\Gamma,\Gamma'\vdash Q;C_2}{\Delta;\Gamma,\Gamma'\vdash P\,|\,Q;C_1\otimes C_2}\ [\text{TPAR}]$$

   Let $C = C_1 \otimes C_2$.

   By applying [TPAR] we obtain $\Delta;\Gamma,\Gamma'\vdash P;C_1$, respectively $\Delta;\Gamma,\Gamma'\vdash Q;C_2$. By applying the inductive hypothesis on these two we obtain $\Delta;\Gamma\vdash P;C_1$ and $\Delta;\Gamma\vdash Q;C_2$. By applying [TPAR] we obtain our conclusion:

   $$\frac{\Delta;\Gamma\vdash P;C_1 \quad \Delta;\Gamma\vdash Q;C_2}{\Delta;\Gamma\vdash P\,|\,Q;C_1\otimes C_2}\ [\text{TPAR}]$$

- case $P = (\nu\, s)P$

1. $\Delta,\Delta';\Gamma\vdash (\nu\, s)P;\ C$

   [TRES]

   $$\frac{\Delta,\Delta';\Gamma,\Gamma'\vdash P;C\otimes C' \quad (\Gamma' = \{\mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho_{\mathbf{p}}), \rho_{\mathbf{p}} : \{\rho_{\mathbf{p}}\mapsto S_{\mathbf{p}}\}\}_{\mathbf{p}\in I}, C' = \otimes_{\mathbf{p}\in I}\{\rho_{\mathbf{p}}\mapsto S_{\mathbf{p}}\})\ \text{complete}}{\Delta,\Delta';\Gamma\vdash (\nu\, s:\Gamma')P;C}$$

   By applying [TRES] we obtain $\Delta,\Delta';\Gamma,\Gamma'\vdash P;C\otimes C'$. By applying the induction hypothesis we obtain: $\Delta;\Gamma,\Gamma'\vdash P;C\otimes C'$. By applying [TRES] we obtain our conclusion:

   [TRES]

   $$\frac{\Delta;\Gamma,\Gamma'\vdash P;C\otimes C' \quad (\Gamma' = \{\mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho_{\mathbf{p}}), \rho_{\mathbf{p}} : \{\rho_{\mathbf{p}}\mapsto S_{\mathbf{p}}\}\}_{\mathbf{p}\in I}, C' = \otimes_{\mathbf{p}\in I}\{\rho_{\mathbf{p}}\mapsto S_{\mathbf{p}}\})\ \text{complete}}{\Delta;\Gamma\vdash (\nu\, s:\Gamma')P;C}$$

2. $\Delta;\Gamma,\Gamma_1\vdash (\nu\, s)P;\ C$

   [TRES]

   $$\frac{\Delta;\Gamma,\Gamma',\Gamma_1\vdash P;C\otimes C' \quad (\Gamma' = \{\mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho_{\mathbf{p}}), \rho_{\mathbf{p}} : \{\rho_{\mathbf{p}}\mapsto S_{\mathbf{p}}\}\}_{\mathbf{p}\in I}, C' = \otimes_{\mathbf{p}\in I}\{\rho_{\mathbf{p}}\mapsto S_{\mathbf{p}}\})\ \text{complete}}{\Delta;\Gamma,\Gamma_1\vdash (\nu\, s:\Gamma')P;C}$$

   By applying [TRES] we obtain $\Delta;\Gamma,\Gamma',\Gamma_1\vdash P;C\otimes C'$ with $(\Gamma' = \{\mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho_{\mathbf{p}}), \rho_{\mathbf{p}} : \{\rho_{\mathbf{p}}\mapsto S_{\mathbf{p}}\}\}_{\mathbf{p}\in I}, C' = \otimes_{\mathbf{p}\in I}\{\rho_{\mathbf{p}}\mapsto S_{\mathbf{p}}\})$ complete. By applying the induction hypothesis we obtain: $\Delta;\Gamma,\Gamma'\vdash P;C\otimes C'$. By applying [TRES] we obtain our conclusion:

[TRES]

$$\frac{\Delta;\Gamma,\Gamma' \vdash P;C\otimes C' \quad (\Gamma' = \{\mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho_{\mathbf{p}}), \rho_{\mathbf{p}} : \{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}\}\}_{\mathbf{p}\in I}, C' = \otimes_{\mathbf{p}\in I}\{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}\}) \text{ complete}}{\Delta;\Gamma \vdash (\nu\, s : \Gamma')P;C}$$

- case $P = \mathtt{c}[\mathbf{p}] \oplus \langle l(v)\rangle.P$

  1. $\Delta,\Delta';\Gamma \vdash \mathtt{c}[\mathbf{p}] \oplus \langle l(v)\rangle.P;\ C$

  $$\frac{\Gamma \vdash v : U;C \quad \Delta,\Delta';\Gamma \vdash P;C'\otimes\{\rho \mapsto S_j\} \quad \mathtt{c} : \mathtt{tr}(\rho), \rho : \{\rho \mapsto S_j\} \in \Gamma \quad j \in I}{\Delta,\Delta';\Gamma \vdash \mathtt{c}[\mathbf{p}] \oplus \langle l_j(v)\rangle.P;C\otimes C'\otimes\{\rho \mapsto \mathbf{p} \oplus_{i\in I} !l_i(U_i).S_i\}}\ [\text{TSEL}]$$

  By applying [TSEL] we obtain $\Delta,\Delta';\Gamma \vdash P;C'\otimes\{\rho \mapsto S_j\}$. By applying the induction hypothesis we obtain:$\Delta;\Gamma \vdash P;C'\otimes\{\rho \mapsto S_j\}$. By applying [TSEL] we obtain our conclusion:

  $$\frac{\Gamma \vdash v : U;C \quad \Delta;\Gamma \vdash P;C'\otimes\{\rho \mapsto S_j\} \quad \mathtt{c} : \mathtt{tr}(\rho), \rho : \{\rho \mapsto S_j\} \in \Gamma \quad j \in I}{\Delta;\Gamma \vdash \mathtt{c}[\mathbf{p}] \oplus \langle l_j(v)\rangle.P;C\otimes C'\otimes\{\rho \mapsto \mathbf{p} \oplus_{i\in I} !l_i(U_i).S_i\}}\ [\text{TSEL}]$$

  2. $\Delta;\Gamma,\Gamma' \vdash \mathtt{c}[\mathbf{p}] \oplus \langle l(v)\rangle.P;\ C$

  $$\frac{\Gamma,\Gamma' \vdash v : U;C \quad \Delta;\Gamma,\Gamma' \vdash P;C'\otimes\{\rho \mapsto S_j\} \quad \mathtt{c} : \mathtt{tr}(\rho), \rho : \{\rho \mapsto S_j\} \in \Gamma \quad j \in I}{\Delta;\Gamma,\Gamma' \vdash \mathtt{c}[\mathbf{p}] \oplus \langle l_j(v)\rangle.P;C\otimes C'\otimes\{\rho \mapsto \mathbf{p} \oplus_{i\in I} !l_i(U_i).S_i\}}\ [\text{TSEL}]$$

  By applying [TSEL] we obtain $\Gamma,\Gamma' \vdash v : U;C$ and $\Delta;\Gamma,\Gamma' \vdash P;C'\otimes\{\rho \mapsto S_j\}$. By applying the induction hypothesis we obtain $\Gamma \vdash v : U;C$ and $\Delta;\Gamma \vdash P;C'\otimes\{\rho \mapsto S_j\}$. By applying [TSEL] we obtain our conclusion:

  $$\frac{\Gamma \vdash v : U;C \quad \Delta;\Gamma \vdash P;C'\otimes\{\rho \mapsto S_j\} \quad \mathtt{c} : \mathtt{tr}(\rho), \rho : \{\rho \mapsto S_j\} \in \Gamma \quad j \in I}{\Delta;\Gamma \vdash \mathtt{c}[\mathbf{p}] \oplus \langle l_j(v)\rangle.P;C\otimes C'\otimes\{\rho \mapsto \mathbf{p} \oplus_{i\in I} !l_i(U_i).S_i\}}\ [\text{TSEL}]$$

- case $P = \mathtt{c}[\mathbf{p}]\&_{i\in I}\{l_i(x_i).P_i\}$

  1. $\Delta,\Delta';\Gamma \vdash \mathtt{c}[\mathbf{p}]\&_{i\in I}\{l_i(x_i).P_i\};\ C$

  $$\frac{\Delta,\Delta';\Gamma,x_i : U_i \vdash P_i;C\otimes C_i\otimes\{\rho \mapsto S_i\} \quad \mathtt{c} : \mathtt{tr}(\rho), \rho : \{\rho \mapsto S_i\} \in \Gamma \quad \forall i \in I}{\Delta,\Delta';\Gamma \vdash \mathtt{c}[\mathbf{p}]\&_{i\in I}\{l_i(x_i).P_i\};C\otimes\{\rho \mapsto \mathbf{p}\ \&_{i\in I}?l_i(U_i).S_i\}}\ [\text{TBR}]$$

  By applying [TBR] we obtain $\Delta,\Delta';\Gamma,x_i : U_i \vdash P_i;C\otimes C_i\otimes\{\rho \mapsto S_i\}$. By applying the induction hypothesis we obtain $\Delta;\Gamma,x_i : U_i \vdash P_i;C\otimes C_i\otimes\{\rho \mapsto S_i\}$. By applying [TBR] we obtain:

  $$\frac{\Delta;\Gamma,x_i : U_i \vdash P_i;C\otimes C_i\otimes\{\rho \mapsto S_i\} \quad \mathtt{c} : \mathtt{tr}(\rho), \rho : \{\rho \mapsto S_i\} \in \Gamma \quad \forall i \in I}{\Delta;\Gamma \vdash \mathtt{c}[\mathbf{p}]\&_{i\in I}\{l_i(x_i).P_i\};C\otimes\{\rho \mapsto \mathbf{p}\ \&_{i\in I}?l_i(U_i).S_i\}}\ [\text{TBR}]$$

  2. $\Delta;\Gamma,\Gamma' \vdash \mathtt{c}[\mathbf{p}]\&_{i\in I}\{l_i(x_i).P_i\};\ C$

  $$\frac{\Delta;\Gamma,x_i : U_i,\Gamma' \vdash P_i;C\otimes C_i\otimes\{\rho \mapsto S_i\} \quad \mathtt{c} : \mathtt{tr}(\rho), \rho : \{\rho \mapsto S_i\} \in \Gamma,\Gamma' \quad \forall i \in I}{\Delta;\Gamma,\Gamma' \vdash \mathtt{c}[\mathbf{p}]\&_{i\in I}\{l_i(x_i).P_i\};C\otimes\{\rho \mapsto \mathbf{p}\ \&_{i\in I}?l_i(U_i).S_i\}}\ [\text{TBR}]$$

By applying [TBR] we obtain $\Delta;\Gamma,x_i:U_i,\Gamma' \vdash P_i;C\otimes C_i\otimes\{\rho\mapsto S_i\}$. By applying the induction hypothesis we obtain $\Delta;\Gamma,x_i:U_i\vdash P_i;C\otimes C_i\otimes\{\rho\mapsto S_i\}$. By applying [TBR] we obtain:

$$\frac{\Delta;\Gamma,x_i:U_i\vdash P_i;C\otimes C_i\otimes\{\rho\mapsto S_i\} \quad \mathsf{c}:\mathtt{tr}(\rho),\rho:\{\rho\mapsto S_i\}\in\Gamma \quad \forall i\in I}{\Delta;\Gamma\vdash \mathsf{c}[\mathbf{p}]\&_{i\in I}\{l_i(x_i).P_i\};C\otimes\{\rho\mapsto \mathbf{p}\ \&_{i\in I}?l_i(U_i).S_i\}}\ [\text{TBR}]$$

- case $P=\mathsf{c}[\mathbf{p}]\oplus\langle l(\mathsf{pack}(\rho,\mathsf{s}[\mathbf{q}]))\rangle.P$

  1. $\Delta,\Delta';\Gamma\vdash \mathsf{c}[\mathbf{p}]\oplus\langle l(\mathsf{pack}(\rho,\mathsf{s}[\mathbf{q}]))\rangle.P;\ C$
  [TSELP]
  $$\frac{\Gamma\vdash v:\mathtt{tr}(\rho');\varnothing \quad \Delta,\Delta';\Gamma\vdash P;C\otimes\{\rho\mapsto S_j,\rho'\mapsto U\} \quad \mathsf{c}:\mathtt{tr}(\rho),\rho:\{\rho\mapsto S_j\}\in\Gamma \quad j\in I}{\Delta,\Delta';\Gamma\vdash \mathsf{c}[\mathbf{p}]\oplus\langle l_j(\mathsf{pack}(\rho',v))\rangle.P;C\otimes\{\rho\mapsto \mathbf{p}\oplus_{i\in I}!l_i(\exists[\rho'|\{\rho'\mapsto U\}].\mathtt{tr}(\rho')).S_i,\rho'\mapsto U\}}$$
  By applying [TSELP] we obtain $\Delta,\Delta';\Gamma\vdash P;C\otimes\{\rho\mapsto S_j,\rho'\mapsto U\}$. By applying the induction hypothesis we obtain $\Delta;\Gamma\vdash P;C\otimes\{\rho\mapsto S_j,\rho'\mapsto U\}$. By applying [TSELP] we obtain our conclusion:
  [TSELP]
  $$\frac{\Gamma\vdash v:\mathtt{tr}(\rho');\varnothing \quad \Delta;\Gamma\vdash P;C\otimes\{\rho\mapsto S_j,\rho'\mapsto U\} \quad \mathsf{c}:\mathtt{tr}(\rho),\rho:\{\rho\mapsto S_j\}\in\Gamma \quad j\in I}{\Delta;\Gamma\vdash \mathsf{c}[\mathbf{p}]\oplus\langle l_j(\mathsf{pack}(\rho',v))\rangle.P;C\otimes\{\rho\mapsto \mathbf{p}\oplus_{i\in I}!l_i(\exists[\rho'|\{\rho'\mapsto U\}].\mathtt{tr}(\rho')).S_i,\rho'\mapsto U\}}$$

  2. $\Delta;\Gamma,\Gamma'\vdash \mathsf{c}[\mathbf{p}]\oplus\langle l(\mathsf{pack}(\rho,\mathsf{s}[\mathbf{q}]))\rangle.P;\ C$
  [TSELP]
  $$\frac{\Gamma,\Gamma'\vdash v:\mathtt{tr}(\rho');\varnothing \quad \Delta;\Gamma,\Gamma'\vdash P;C\otimes\{\rho\mapsto S_j,\rho'\mapsto U\} \quad \mathsf{c}:\mathtt{tr}(\rho),\rho:\{\rho\mapsto S_j\}\in\Gamma,\Gamma' \quad j\in I}{\Delta;\Gamma,\Gamma'\vdash \mathsf{c}[\mathbf{p}]\oplus\langle l_j(\mathsf{pack}(\rho',v))\rangle.P;C\otimes\{\rho\mapsto \mathbf{p}\oplus_{i\in I}!l_i(\exists[\rho'|\{\rho'\mapsto U\}].\mathtt{tr}(\rho')).S_i,\rho'\mapsto U\}}$$
  By applying [TSELP] we obtain $\Delta;\Gamma,\Gamma'\vdash P;C\otimes\{\rho\mapsto S_j,\rho'\mapsto U\}$ and $\Gamma,\Gamma'\vdash v:\mathtt{tr}(\rho');\varnothing$. By applying the induction hypothesis we obtain $\Delta;\Gamma\vdash P;C\otimes\{\rho\mapsto S_j,\rho'\mapsto U\}$ and $\Gamma\vdash v:\mathtt{tr}(\rho');\varnothing$. By applying [TSELP] we obtain our conclusion:
  [TSELP]
  $$\frac{\Gamma\vdash v:\mathtt{tr}(\rho');\varnothing \quad \Delta;\Gamma\vdash P;C\otimes\{\rho\mapsto S_j,\rho'\mapsto U\} \quad \mathsf{c}:\mathtt{tr}(\rho),\rho:\{\rho\mapsto S_j\}\in\Gamma \quad j\in I}{\Delta;\Gamma\vdash \mathsf{c}[\mathbf{p}]\oplus\langle l_j(\mathsf{pack}(\rho',v))\rangle.P;C\otimes\{\rho\mapsto \mathbf{p}\oplus_{i\in I}!l_i(\exists[\rho'|\{\rho'\mapsto U\}].\mathtt{tr}(\rho')).S_i,\rho'\mapsto U\}}$$

- case $P=\mathsf{c}[\mathbf{p}]\&_{i\in I}\{l_i(\mathsf{pack}(\rho_i,\mathtt{s}_i[\mathbf{q}])).P_i\}$

  1. $\Delta,\Delta';\Gamma\vdash \mathsf{c}[\mathbf{p}]\&_{i\in I}\{l_i(\mathsf{pack}(\rho_i,\mathtt{s}_i[\mathbf{q}])).P_i\};\ C$
  [TBRP]
  $$\frac{\Delta,\Delta';\Gamma,\mathtt{v}_i:\mathtt{tr}(\rho_i),\rho_i:\{\rho_i\mapsto U_i\}\vdash P_i;C\otimes\{\rho\mapsto S_i\} \quad \forall i\in I \quad \mathsf{c}:\mathtt{tr}(\rho),\rho:\{\rho\mapsto S_i\}\in\Gamma}{\Delta,\Delta';\Gamma\vdash \mathsf{c}[\mathbf{p}]\&_{i\in I}\{l_i(\mathsf{pack}(\rho_i,\mathtt{v}_i)).P_i\};C\otimes\{\rho\mapsto \mathbf{p}\ \&_{i\in I}?l_i(\exists[\rho_i|\{\rho_i\mapsto U_i\}].\mathtt{tr}(\rho_i)).S_i\}}$$
  By applying [TBRP] we obtain $\Delta,\Delta';\Gamma,\mathtt{v}_i:\mathtt{tr}(\rho_i),\rho_i:\{\rho_i\mapsto U_i\}\vdash P_i;C\otimes\{\rho\mapsto S_i\}$. By applying the induction hypothesis we obtain $\Delta;\Gamma,\mathtt{v}_i:\mathtt{tr}(\rho_i),\rho_i:\{\rho_i\mapsto U_i\}\vdash P_i;C\otimes\{\rho\mapsto S_i\}$. By applying [TBRP] we obtain our conclusion:
  [TBRP]
  $$\frac{\Delta;\Gamma,\mathtt{v}_i:\mathtt{tr}(\rho_i),\rho_i:\{\rho_i\mapsto U_i\}\vdash P_i;C\otimes\{\rho\mapsto S_i\} \quad \forall i\in I \quad \mathsf{c}:\mathtt{tr}(\rho),\rho:\{\rho\mapsto S_i\}\in\Gamma}{\Delta;\Gamma\vdash \mathsf{c}[\mathbf{p}]\&_{i\in I}\{l_i(\mathsf{pack}(\rho_i,\mathtt{v}_i)).P_i\};C\otimes\{\rho\mapsto \mathbf{p}\ \&_{i\in I}?l_i(\exists[\rho_i|\{\rho_i\mapsto U_i\}].\mathtt{tr}(\rho_i)).S_i\}}$$

2. $\Delta;\Gamma,\Gamma' \vdash \mathtt{c}[\mathbf{p}]\&_{i\in I}\{l_i(\mathsf{pack}(\rho_i,\mathtt{s_i}[\mathbf{q}])).P_i\};\ C$
[TBRP]
$$\frac{\Delta;\Gamma,\Gamma',\mathtt{v_i}:\mathtt{tr}(\rho_i),\rho_i:\{\rho_i\mapsto U_i\}\vdash P_i;C\otimes\{\rho\mapsto S_i\}\quad \forall i\in I\quad \mathtt{c}:\mathtt{tr}(\rho),\rho:\{\rho\mapsto S_i\}\in\Gamma,\Gamma'}{\Delta;\Gamma,\Gamma'\vdash \mathtt{c}[\mathbf{p}]\&_{i\in I}\{l_i(\mathsf{pack}(\rho_i,\mathtt{v_i})).P_i\};C\otimes\{\rho\mapsto \mathbf{p}\ \&_{i\in I}?l_i(\exists[\rho_i|\{\rho_i\mapsto U_i\}].\mathtt{tr}(\rho_i)).S_i\}}$$
By applying [TBRP] we obtain $\Delta;\Gamma,\Gamma',\mathtt{v_i}:\mathtt{tr}(\rho_i),\rho_i:\{\rho_i\mapsto U_i\}\vdash P_i;C\otimes\{\rho\mapsto S_i\}$. By applying the induction hypothesis we obtain $\Delta;\Gamma,\mathtt{v_i}:\mathtt{tr}(\rho_i),\rho_i:\{\rho_i\mapsto U_i\}\vdash P_i;C\otimes\{\rho\mapsto S_i\}$. By applying [TBRP] we obtain our conclusion:
[TBRP]
$$\frac{\Delta;\Gamma,\mathtt{v_i}:\mathtt{tr}(\rho_i),\rho_i:\{\rho_i\mapsto U_i\}\vdash P_i;C\otimes\{\rho\mapsto S_i\}\quad \forall i\in I\quad \mathtt{c}:\mathtt{tr}(\rho),\rho:\{\rho\mapsto S_i\}\in\Gamma}{\Delta;\Gamma\vdash \mathtt{c}[\mathbf{p}]\&_{i\in I}\{l_i(\mathsf{pack}(\rho_i,\mathtt{v_i})).P_i\};C\otimes\{\rho\mapsto \mathbf{p}\ \&_{i\in I}?l_i(\exists[\rho_i|\{\rho_i\mapsto U_i\}].\mathtt{tr}(\rho_i)).S_i\}}$$

- **def** D **in** P

1. $\Delta,\Delta';\Gamma\vdash$ **def** D **in** $P;\ C$
$$\frac{\Delta,X:\tilde{U},\Delta';\tilde{x}:\tilde{U}\vdash P;\tilde{C}\qquad \Delta,X:\tilde{U},\Delta';\Gamma\vdash Q;C}{\Delta,\Delta';\Gamma\vdash \textbf{def}\ X\langle\tilde{x}:\tilde{U}\rangle=P;\tilde{C}\ \textbf{in}\ Q;C}\ [\textsc{TDef}]$$
By applying [TDEF] we obtain $\Delta,X:\tilde{U},\Delta';\tilde{x}:\tilde{U}\vdash P;\tilde{C}$ and $\Delta,X:\tilde{U},\Delta';\Gamma\vdash Q;C$. By applying the induction hypothesis we obtain $\Delta,X:\tilde{U};\tilde{x}:\tilde{U}\vdash P;\tilde{C}$ and $\Delta,X:\tilde{U};\Gamma\vdash Q;C$. By applying [TDEF] we obtain our conclusion:
$$\frac{\Delta,X:\tilde{U};\tilde{x}:\tilde{U}\vdash P;\tilde{C}\qquad \Delta,X:\tilde{U};\Gamma\vdash Q;C}{\Delta;\Gamma\vdash \textbf{def}\ X\langle\tilde{x}:\tilde{U}\rangle=P;\tilde{C}\ \textbf{in}\ Q;C}\ [\textsc{TDef}]$$

2. $\Delta;\Gamma,\Gamma'\vdash$ **def** D **in** $P;\ C$
$$\frac{\Delta,X:\tilde{U};\tilde{x}:\tilde{U}\vdash P;\tilde{C}\qquad \Delta,X:\tilde{U};\Gamma,\Gamma'\vdash Q;C}{\Delta,\Delta';\Gamma,\Gamma'\vdash \textbf{def}\ X\langle\tilde{x}:\tilde{U}\rangle=P;\tilde{C}\ \textbf{in}\ Q;C}\ [\textsc{TDef}]$$
By applying [TDEF] we obtain $\Delta,X:\tilde{U};\Gamma,\Gamma'\vdash Q;C$. By applying the induction hypothesis we obtain $\Delta,X:\tilde{U};\Gamma\vdash Q;C$. By applying [TDEF] we obtain our conclusion:
$$\frac{\Delta,X:\tilde{U};\tilde{x}:\tilde{U}\vdash P;\tilde{C}\qquad \Delta,X:\tilde{U};\Gamma\vdash Q;C}{\Delta;\Gamma\vdash \textbf{def}\ X\langle\tilde{x}:\tilde{U}\rangle=P;\tilde{C}\ \textbf{in}\ Q;C}\ [\textsc{TDef}]$$

- $P=X\langle\tilde{x}\rangle$

1. $\Delta,\Delta';\Gamma\vdash X\langle\tilde{x}\rangle;\ C$
By applying [TCALL] we obtain our conclusion:
[TCALL]
$$\frac{\forall i\in\{1..n\}\quad \Gamma\vdash v_i:U_i;C_i}{\Delta,X:U_1,\ldots,U_n;\Gamma\vdash X\langle v_1,\ldots,v_n\rangle;C_1\otimes\ldots\otimes C_n}$$

2. $\Delta;\Gamma,\Gamma'\vdash X\langle\tilde{x}\rangle;\ C$
[TCALL]
$$\frac{\forall i\in\{1..n\}\quad \Gamma,\Gamma'\vdash v_i:U_i;C_i}{\Delta,X:U_1,\ldots,U_n;\Gamma,\Gamma'\vdash X\langle v_1,\ldots,v_n\rangle;C_1\otimes\ldots\otimes C_n}$$
By applying [TCALL] we obtain $\Gamma,\Gamma'\vdash v_i:U_i;C_i$. By applying the induction hypothesis we obtain $\Gamma\vdash v_i:U_i;C_i$. By

applying [TCALL] we obtain our conclusion:

$$\frac{\forall i \in \{1..n\} \quad \Gamma \vdash v_i : U_i; C_i}{\Delta, X : U_1, \ldots, U_n; \Gamma \vdash X\langle v_1, \ldots, v_n \rangle; C_1 \otimes \ldots \otimes C_n} [\text{TCALL}]$$

□

Proposition A.1.1. *If* $\Delta; \Gamma \vdash P; C$ *and there exists* C' *such that* C' *terminated and* C *and* C' *are disjoint then* $\Delta; \Gamma \vdash P; C \otimes C'$.

Proof of Proposition A.1.1. By induction on typing derivations, with a case analysis on the last rule applied. □

Proposition 4.5.3. *For all multiparty session processes* P, P' *if* $\Delta; \Gamma \vdash P; C$ *and* $P \equiv P'$*, then* $\Delta; \Gamma \vdash P'; C$.

Proof of Proposition 4.5.3. The proof proceeds by induction on the structural congruence $\equiv$.

- $P \mid \mathbf{0} \equiv P$ Left to right we have $\Delta;\ \Gamma \vdash P \mid \mathbf{0};\ C$ and $P \mid \mathbf{0} \equiv P$.

  $$\frac{\Delta;\ \Gamma \vdash P;\ C_1 \qquad \dfrac{C_2 \text{ terminated}}{\Delta;\ \Gamma \vdash \mathbf{0};\ C_2}\,[\text{TINACT}]}{\Delta;\ \Gamma \vdash P \mid \mathbf{0};\ C = C_1 \otimes C_2}\,[\text{TPAR}]$$

  Since C_2 terminated and C_2 and C_1 are disjoint then by Proposition A.1.1 we have: $\Delta;\ \Gamma \vdash P;\ C_1 \otimes C_2$.

  Right to left we have $\Delta;\ \Gamma \vdash P;\ C$ and $P \mid \mathbf{0} \equiv P$. From [TINACT] we know: $\Delta;\ \Gamma \vdash \mathbf{0}; \varnothing$.

  $$\frac{\Delta;\ \Gamma \vdash P;\ C \qquad \Delta;\ \Gamma \vdash \mathbf{0}; \varnothing}{\Delta;\ \Gamma \vdash P \mid \mathbf{0};\ C}\,[\text{TPAR}]$$

- $P \mid Q \equiv Q \mid P$ Left to right we have: $\Delta;\ \Gamma \vdash P \mid Q;\ C_1 \otimes C_2$ and $P \mid Q \equiv Q \mid P$.

  $$\frac{\Delta;\ \Gamma \vdash P;\ C_1 \qquad \Delta;\ \Gamma \vdash Q;\ C_2}{\Delta;\ \Gamma \vdash P \mid Q;\ C_1 \otimes C_2}\,[\text{TPAR}]$$

  By applying [TPAR] to the two premises we get:

  $$\frac{\Delta;\ \Gamma \vdash Q;\ C_2 \qquad \Delta;\ \Gamma \vdash P;\ C_1}{\Delta;\ \Gamma \vdash Q \mid P;\ C_2 \otimes C_1}\,[\text{TPAR}]$$

  $C_1 \otimes C_2 = C_2 \otimes C_1$.

  Right to left we have $\Delta;\ \Gamma \vdash Q \mid P;\ C_2 \otimes C_1$ and $P \mid Q \equiv Q \mid P$.

  $$\frac{\Delta;\ \Gamma \vdash Q;\ C_2 \qquad \Delta;\ \Gamma \vdash P;\ C_1}{\Delta;\ \Gamma \vdash Q \mid P;\ C_2 \otimes C_1}\,[\text{TPAR}]$$

By applying [TPAR] to the two premises we get:

$$\frac{\Delta;\ \Gamma \vdash P;\ C_1 \qquad \Delta;\ \Gamma \vdash Q;\ C_2}{\Delta;\ \Gamma \vdash P \mid Q;\ C_1 \otimes C_2}\ [\text{TPAR}]$$

$C_2 \otimes C_1 = C_1 \otimes C_2$.

- $(P \mid Q) \mid R \equiv P(Q \mid R)$ Left to right we have: $\Delta;\ \Gamma \vdash (P \mid Q) \mid R; (C_1 \otimes C_2) \otimes C_3$ and $(P \mid Q) \mid R \equiv P(Q \mid R)$.

$$\frac{\dfrac{\Delta;\ \Gamma \vdash P;\ C_1 \qquad \Delta;\ \Gamma \vdash Q;\ C_2}{\Delta;\ \Gamma \vdash P \mid Q;\ C_1 \otimes C_2}\ [\text{TPAR}] \qquad \Delta;\ \Gamma \vdash R; C_3}{\Delta;\ \Gamma \vdash (P \mid Q) \mid R; (C_1 \otimes C_2) \otimes C_3}\ [\text{TPAR}]$$

By applying [TPAR] to the three premises we get:

$$\frac{\Delta;\ \Gamma \vdash P;\ C_1 \qquad \dfrac{\Delta;\ \Gamma \vdash Q;\ C_2 \qquad \Delta;\ \Gamma \vdash R; C_3}{\Delta;\ \Gamma \vdash Q \mid R;\ C_2 \otimes C_3}\ [\text{TPAR}]}{\Delta;\ \Gamma \vdash P(Q \mid R);\ C_1 \otimes (C_2 \otimes C_3)}\ [\text{TPAR}]$$

$(C_1 \otimes C_2) \otimes C_3 = C_1 \otimes (C_2 \otimes C_3)$.

Right to left we have $\Delta;\ \Gamma \vdash P(Q \mid R);\ C_1 \otimes (C_2 \otimes C_3)$.

$$\frac{\Delta;\ \Gamma \vdash P;\ C_1 \qquad \dfrac{\Delta;\ \Gamma \vdash Q;\ C_2 \qquad \Delta;\ \Gamma \vdash R; C_3}{\Delta;\ \Gamma \vdash Q \mid R;\ C_2 \otimes C_3}\ [\text{TPAR}]}{\Delta;\ \Gamma \vdash P(Q \mid R);\ C_1 \otimes (C_2 \otimes C_3)}\ [\text{TPAR}]$$

By applying [TPAR] to the three premises we get:

$$\frac{\dfrac{\Delta;\ \Gamma \vdash P;\ C_1 \qquad \Delta;\ \Gamma \vdash Q;\ C_2}{\Delta;\ \Gamma \vdash P \mid Q;\ C_1 \otimes C_2}\ [\text{TPAR}] \qquad \Delta;\ \Gamma \vdash R; C_3}{\Delta;\ \Gamma \vdash (P \mid Q) \mid R; (C_1 \otimes C_2) \otimes C_3}\ [\text{TPAR}]$$

$C_1 \otimes (C_2 \otimes C_3) = (C_1 \otimes C_2) \otimes C_3$.

- $(\nu\, s)\mathbf{0} \equiv \mathbf{0}$ Left to right we have: $\Delta;\ \Gamma \vdash (\nu\, s)\mathbf{0};\ C$ and $(\nu\, s)\mathbf{0} \equiv \mathbf{0}$. By inversion on [TRES] and [TINACT] we obtain C terminated.

  From [TINACT] we have $\Delta;\ \Gamma \vdash \mathbf{0};\ C$.

  Right to left we have $\Delta;\ \Gamma \vdash \mathbf{0};\ C$ and $(\nu\, s)\mathbf{0} \equiv \mathbf{0}$. From [TINACT] we have C terminated.

$$\frac{(\Gamma' = \{\mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho_{\mathbf{p}}), \rho_{\mathbf{p}} : \{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}\}\}_{\mathbf{p} \in I}, C' = \otimes_{\mathbf{p} \in I}\{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}\}) \qquad \dfrac{C \otimes C' \text{ terminated}}{\Delta;\ \Gamma, \Gamma' \vdash \mathbf{0};\ C \otimes C'}\ [\text{TINACT}]}{\Delta;\ \Gamma \vdash (\nu\, s : \Gamma')\mathbf{0};\ C}\ [\text{TRES}]$$

- $(\nu\, s)(\nu\, s')P \equiv (\nu\, s')(\nu\, s)P$

Left to right: We construct the derivation for $(\nu\, s)(\nu\, s')P$.

$$\dfrac{\dfrac{\Delta;\ \Gamma,\Gamma',\Gamma'' \vdash P;\ C\otimes C'\otimes C''}{\Gamma'' = \{\mathtt{s}'[\mathbf{p}'_i] : \mathtt{tr}(\rho'_i), \rho_{\mathbf{p}'} : \{\rho_{\mathbf{p}'} \mapsto S_{p'}\}\}_{i\in I}, C'' = \otimes_{\mathbf{p}'\in I}\{\rho_{\mathbf{p}'} \mapsto S_{p'}\}}\ [\text{TRES}]}{\Delta;\ \Gamma,\Gamma' \vdash (\nu\, s' : \Gamma'')P;\ C\otimes C'} \quad \vdots \quad (\Gamma' = \{\mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho_{\mathbf{p}}), \rho_{\mathbf{p}} : \{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}\}\}_{\mathbf{p}\in I}, C' = \otimes_{\mathbf{p}\in I}\{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}\})}{\Delta;\ \Gamma \vdash (\nu\, s : \Gamma')(\nu\, s' : \Gamma'')P;\ C}\ [\text{TRES}]$$

By rearranging:

$$\dfrac{\dfrac{\Delta;\ \Gamma,\Gamma',\Gamma'' \vdash P;\ C\otimes C'\otimes C''}{(\Gamma' = \{\mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho_{\mathbf{p}}), \rho_{\mathbf{p}} : \{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}\}\}_{\mathbf{p}\in I}, C' = \otimes_{\mathbf{p}\in I}\{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}\})}\ [\text{TRES}]}{\Delta;\ \Gamma,\Gamma'' \vdash (\nu\, s : \Gamma')P;\ C\otimes C''} \quad \vdots \quad \Gamma'' = \{\mathtt{s}'[\mathbf{p}'_i] : \mathtt{tr}(\rho'_i), \rho_{\mathbf{p}'} : \{\rho_{\mathbf{p}'} \mapsto S_{p'}\}\}_{i\in I}, C'' = \otimes_{\mathbf{p}'\in I}\{\rho_{\mathbf{p}'} \mapsto S_{p'}\}}{\Delta;\ \Gamma \vdash (\nu\, s' : \Gamma'')(\nu\, s : \Gamma')P;\ C}\ [\text{TRES}]$$

Right to left:

$$\dfrac{\dfrac{\Delta;\ \Gamma,\Gamma',\Gamma'' \vdash P;\ C\otimes C'\otimes C''}{(\Gamma' = \{\mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho_{\mathbf{p}}), \rho_{\mathbf{p}} : \{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}\}\}_{\mathbf{p}\in I}, C' = \otimes_{\mathbf{p}\in I}\{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}\})}\ [\text{TRES}]}{\Delta;\ \Gamma,\Gamma'' \vdash (\nu\, s : \Gamma')P;\ C\otimes C''} \quad \vdots \quad \Gamma'' = \{\mathtt{s}'[\mathbf{p}'] : \mathtt{tr}(\rho'_{\mathbf{p}'}), \rho_{\mathbf{p}'} : \{\rho_{\mathbf{p}'} \mapsto S_{p'}\}\}_{\mathbf{p}'\in I}, C'' \otimes_{\mathbf{p}'\in I}\{\rho_{\mathbf{p}'} \mapsto S_{p'}\}}{\Delta;\ \Gamma \vdash (\nu\, s' : \Gamma'')(\nu\, s : \Gamma')P;\ C}\ [\text{TRES}]$$

By rearranging the premises:

$$\dfrac{\dfrac{\Delta;\ \Gamma,\Gamma',\Gamma'' \vdash P;\ C\otimes C'\otimes C''}{\Gamma'' = \{\mathtt{s}'[\mathbf{p}'_i] : \mathtt{tr}(\rho'_i), \rho_{\mathbf{p}'} : \{\rho_{\mathbf{p}'} \mapsto S_{p'}\}\}_{i\in I}, C'' = \otimes_{\mathbf{p}'\in I}\{\rho_{\mathbf{p}'} \mapsto S_{p'}\}}\ [\text{TRES}]}{\Delta;\ \Gamma,\Gamma' \vdash (\nu\, s' : \Gamma'')P;\ C\otimes C'} \quad \vdots \quad (\Gamma' = \{\mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho_{\mathbf{p}}), \rho_{\mathbf{p}} : \{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}\}\}_{\mathbf{p}\in I}, C' = \otimes_{\mathbf{p}\in I}\{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}\})}{\Delta;\ \Gamma \vdash (\nu\, s : \Gamma')(\nu\, s' : \Gamma'')P;\ C}\ [\text{TRES}]$$

- $(\nu\, s)P \,|\, Q \equiv (\nu\, s)(P \,|\, Q) \quad \text{if } s \notin fc(Q)$

Left to right:

$$\dfrac{\dfrac{\Delta;\ \Gamma,\Gamma' \vdash P;\ C_1 \otimes C' \qquad (\Gamma' = \{\mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho_{\mathbf{p}}), \rho_{\mathbf{p}} : \{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}\}\}_{\mathbf{p}\in I}, C' = \otimes_{\mathbf{p}\in I}\{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}\})}{\Delta;\ \Gamma \vdash (\nu\, s : \Gamma')P;\ C_1}\ [\text{TRES}] \qquad \Delta;\ \Gamma \vdash Q;\ C_2}{\Delta;\ \Gamma \vdash (\nu\, s : \Gamma')P \mid Q;\ C_1 \otimes C_2}\ [\text{TPAR}]$$

By Proposition 4.5.1, $\Delta;\ \Gamma \vdash Q;\ C_2$ with Γ' we obtain $\Delta;\ \Gamma,\Gamma' \vdash Q;\ C_2$.

$$\dfrac{\dfrac{\Delta;\ \Gamma,\Gamma' \vdash P;\ C_1 \otimes C' \qquad \Delta;\ \Gamma,\Gamma' \vdash Q;\ C_2}{\Delta;\ \Gamma,\Gamma' \vdash P \mid Q;\ C_1 \otimes C_2 \otimes C'}\ [\text{TPAR}] \quad \vdots \quad (\Gamma' = \{\mathtt{s}[\mathbf{p}] : \mathtt{tr}(\rho_{\mathbf{p}}), \rho_{\mathbf{p}} : \{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}\}\}_{\mathbf{p}\in I}, C' = \otimes_{\mathbf{p}\in I}\{\rho_{\mathbf{p}} \mapsto S_{\mathbf{p}}\})}{\Delta;\ \Gamma \vdash (\nu\, s : \Gamma')(P \mid Q);\ C_1 \otimes C_2}\ [\text{TRES}]$$

- **def** $X\langle \tilde{x} : \tilde{U}\rangle = Q_X$ **in** $\mathbf{0} \equiv \mathbf{0}$

  Left to right: From the induction hypothesis we have $\Delta;\ \Gamma \vdash$ **def** $X\langle \tilde{x} : \tilde{U}\rangle = Q_X$ **in** $\mathbf{0}$; C.

$$\dfrac{\Delta, X : \tilde{U}; \tilde{x} : \tilde{U} \vdash Q_X;\ \tilde{C} \qquad \dfrac{C \text{ terminated}}{\Delta, X : \tilde{U}; \Gamma \vdash \mathbf{0};\ C}\ [\text{TINACT}]}{\Delta;\ \Gamma \vdash \textbf{def}\ X\langle \tilde{x} : \tilde{U}\rangle = Q_X\ \textbf{in}\ \mathbf{0};\ C}\ [\text{TDEF}]$$

  From [TINACT]: $\Delta;\ \Gamma \vdash \mathbf{0}; C'$ and C' terminated. Since both C and C' are terminated then $C = C'$.

  Right to left: From the induction hypothesis we have : $\Delta;\ \Gamma \vdash \mathbf{0};\ C$. From [TINACT] we obtain C terminated. We construct the derivation for process: **def** $X\langle \tilde{x} : \tilde{U}\rangle = Q_X$ **in** $\mathbf{0}$.

$$\dfrac{\Delta, X : \tilde{U}; \tilde{x} : \tilde{U} \vdash Q_X;\ \tilde{C} \qquad \dfrac{C' \text{ terminated}}{\Delta, X : \tilde{U}; \Gamma \vdash \mathbf{0}; C'}\ [\text{TINACT}]}{\Delta;\ \Gamma \vdash \textbf{def}\ X\langle \tilde{x} : \tilde{U}\rangle = Q_X\ \textbf{in}\ \mathbf{0}; C'}\ [\text{TDEF}]$$

  Since both C and C' are terminated then $C = C'$.

- **def** $X\langle \tilde{x} : \tilde{U}\rangle = Q_X$ **in** $(\nu\, s)P \equiv (\nu\, s)($**def** $X\langle \tilde{x} : \tilde{U}\rangle = Q_X$ **in** $P)$ $s \notin fc(Q)$

  Left to right: From the induction hypothesis: $\Delta;\ \Gamma \vdash$ **def** $X\langle \tilde{x} : \tilde{U}\rangle = Q_X$ **in** $(\nu\, s : \Gamma')P$; C. We

construct the derivation for this:

$$\dfrac{\Delta, X:\tilde{U};\tilde{x}:\tilde{U}\vdash Q_X;\ \tilde{C} \qquad \dfrac{\Delta, X:\tilde{U};\Gamma,\Gamma'\vdash P;\ C\otimes C' \quad (\Gamma'=\{\mathtt{s}[\mathbf{p}]:\mathtt{tr}(\rho_{\mathbf{p}}),\rho_{\mathbf{p}}:\{\rho_{\mathbf{p}}\mapsto S_{\mathbf{p}}\}\}_{\mathbf{p}\in I}, C'=\otimes_{\mathbf{p}\in I}\{\rho_{\mathbf{p}}\mapsto S_{\mathbf{p}}\})}{\Delta, X:\tilde{U};\Gamma\vdash(\nu s:\Gamma')P;\ C}\ [\text{TRES}]}{\Delta;\ \Gamma\vdash \mathbf{def}\, X\langle\tilde{x}:\tilde{U}\rangle = Q_X \,\mathbf{in}\, (\nu s:\Gamma')P;\ C}\ [\text{TDEF}]$$

By rearranging the premises:

$$\dfrac{\dfrac{\Delta, X:\tilde{U};\Gamma,\Gamma'\vdash P;\ C\otimes C' \qquad \Delta, X:\tilde{U};\tilde{x}:\tilde{U}\vdash Q_X;\ \tilde{C}}{\Delta;\ \Gamma,\Gamma'\vdash \mathbf{def}\, X\langle\tilde{x}:\tilde{U}\rangle = Q_X \,\mathbf{in}\, P;\ C\otimes C'}\ [\text{TDEF}] \quad (\Gamma'=\{\mathtt{s}[\mathbf{p}]:\mathtt{tr}(\rho_{\mathbf{p}}),\rho_{\mathbf{p}}:\{\rho_{\mathbf{p}}\mapsto S_{\mathbf{p}}\}\}_{\mathbf{p}\in I}, C'=\otimes_{\mathbf{p}\in I}\{\rho_{\mathbf{p}}\mapsto S_{\mathbf{p}}\})}{\Delta;\ \Gamma\vdash(\nu s:\Gamma')\mathbf{def}\, X\langle\tilde{x}:\tilde{U}\rangle = Q_X \,\mathbf{in}\, P;\ C}\ [\text{TRES}]$$

Right to left:

$$\dfrac{\dfrac{\Delta, X:\tilde{U};\Gamma,\Gamma'\vdash P;\ C\otimes C' \qquad \Delta, X:\tilde{U};\tilde{x}:\tilde{U}\vdash Q_X;\ \tilde{C}}{\Delta;\ \Gamma,\Gamma'\vdash \mathbf{def}\, X\langle\tilde{x}:\tilde{U}\rangle = Q_X \,\mathbf{in}\, P;\ C\otimes C'}\ [\text{TDEF}] \quad (\Gamma'=\{\mathtt{s}[\mathbf{p}]:\mathtt{tr}(\rho_{\mathbf{p}}),\rho_{\mathbf{p}}:\{\rho_{\mathbf{p}}\mapsto S_{\mathbf{p}}\}\}_{\mathbf{p}\in I}, C'=\otimes_{\mathbf{p}\in I}\{\rho_{\mathbf{p}}\mapsto S_{\mathbf{p}}\})}{\Delta;\ \Gamma\vdash(\nu s:\Gamma')\mathbf{def}\, X\langle\tilde{x}:\tilde{U}\rangle = Q_X \,\mathbf{in}\, P;\ C}\ [\text{TRES}]$$

By rearranging the premises:

$$\dfrac{\Delta, X:\tilde{U};\tilde{x}:\tilde{U}\vdash Q_X;\ \tilde{C} \qquad \dfrac{\Delta, X:\tilde{U};\Gamma,\Gamma'\vdash P;\ C\otimes C' \quad (\Gamma'=\{\mathtt{s}[\mathbf{p}]:\mathtt{tr}(\rho_{\mathbf{p}}),\rho_{\mathbf{p}}:\{\rho_{\mathbf{p}}\mapsto S_{\mathbf{p}}\}\}_{\mathbf{p}\in I}, C'=\otimes_{\mathbf{p}\in I}\{\rho_{\mathbf{p}}\mapsto S_{\mathbf{p}}\})}{\Delta, X:\tilde{U};\Gamma\vdash(\nu s:\Gamma')P;\ C}\ [\text{TRES}]}{\Delta;\ \Gamma\vdash \mathbf{def}\, X\langle\tilde{x}:\tilde{U}\rangle = Q_X \,\mathbf{in}\, (\nu s:\Gamma')P;\ C}\ [\text{TDEF}]$$

- $\mathbf{def}\, X\langle\tilde{x}:\tilde{U}\rangle = Q_X \,\mathbf{in}\, P \,|\, Q \equiv (\mathbf{def}\, X\langle\tilde{x}:\tilde{U}\rangle = Q_X \,\mathbf{in}\, P) \,|\, Q$

  Left to right:

  From the induction hypothesis: $\Delta;\ \Gamma\vdash \mathbf{def}\, X\langle\tilde{x}:\tilde{U}\rangle = Q_X \,\mathbf{in}\, P\,|\,Q;\ C$. We construct the derivation for the process:

$$\dfrac{\Delta, X:\tilde{U};\tilde{x}:\tilde{U}\vdash Q_X;\ \tilde{C} \qquad \dfrac{\Delta, X:\tilde{U};\Gamma\vdash P;\ C_1 \qquad \Delta, X:\tilde{U};\Gamma\vdash Q;\ C_2}{\Delta, X:\tilde{U};\Gamma\vdash P\,|\,Q;\ C=C_1\otimes C_2}\ [\text{TPAR}]}{\Delta;\ \Gamma\vdash \mathbf{def}\, X\langle\tilde{x}:\tilde{U}\rangle = Q_X \,\mathbf{in}\, P\,|\,Q;\ C}\ [\text{TDEF}]$$

By rearranging the premises:

$$\dfrac{\dfrac{\Delta, X:\tilde{U};\tilde{x}:\tilde{U}\vdash Q_X;\ \tilde{C} \qquad \Delta, X:\tilde{U};\Gamma\vdash P;\ C_1}{\Delta, X:\tilde{U};\Gamma\vdash \mathbf{def}\, X\langle\tilde{x}:\tilde{U}\rangle = Q_X \,\mathbf{in}\, P;\ C_1}\,[\text{TDEF}] \qquad \Delta, X:\tilde{U};\Gamma\vdash Q;\ C_2}{\Delta;\ \Gamma\vdash (\mathbf{def}\, X\langle\tilde{x}:\tilde{U}\rangle = Q_X \,\mathbf{in}\, P)\,|\,Q;\ C}\,[\text{TPAR}]$$

Right to left:

From the induction hypothesis: $\Delta;\ \Gamma\vdash (\mathbf{def}\, X\langle\tilde{x}:\tilde{U}\rangle = Q_X \,\mathbf{in}\, P)\,|\,Q;\ C$. We construct the derivation for the process:

$$\dfrac{\dfrac{\Delta, X:\tilde{U};\tilde{x}:\tilde{U}\vdash Q_X;\ \tilde{C} \qquad \Delta, X:\tilde{U};\Gamma\vdash P;\ C_1}{\Delta, X:\tilde{U};\Gamma\vdash \mathbf{def}\, X\langle\tilde{x}:\tilde{U}\rangle = Q_X \,\mathbf{in}\, P;\ C_1}\,[\text{TDEF}] \qquad \Delta, X:\tilde{U};\Gamma\vdash Q;\ C_2}{\Delta;\ \Gamma\vdash (\mathbf{def}\, X\langle\tilde{x}:\tilde{U}\rangle = Q_X \,\mathbf{in}\, P)\,|\,Q;\ C}\,[\text{TPAR}]$$

By rearranging the premises:

$$\dfrac{\Delta, X:\tilde{U};\tilde{x}:\tilde{U}\vdash Q_X;\ \tilde{C} \qquad \dfrac{\Delta, X:\tilde{U};\Gamma\vdash P;\ C_1 \qquad \Delta, X:\tilde{U};\Gamma\vdash Q;\ C_2}{\Delta, X:\tilde{U};\Gamma\vdash P\,|\,Q;\ C = C_1\otimes C_2}\,[\text{TPAR}]}{\Delta;\ \Gamma\vdash \mathbf{def}\, X\langle\tilde{x}:\tilde{U}\rangle = Q_X \,\mathbf{in}\, P\,|\,Q;\ C}\,[\text{TDEF}]$$

- $\mathbf{def}\, X\langle\tilde{x}:\tilde{U}\rangle = Q_X \,\mathbf{in}\, \mathbf{def}\, Y\langle\tilde{y}:\tilde{U}\rangle = Q_Y \,\mathbf{in}\, P \equiv \mathbf{def}\, Y\langle\tilde{y}:\tilde{U}\rangle = Q_Y \,\mathbf{in}\, \mathbf{def}\, X\langle\tilde{x}:\tilde{U}\rangle = Q_X \,\mathbf{in}\, P$

  Left to right: From the induction hypothesis: $\Delta;\ \Gamma\vdash \mathbf{def}\, X\langle\tilde{x}:\tilde{U}\rangle = Q_X \,\mathbf{in}\, \mathbf{def}\, Y\langle\tilde{y}:\tilde{U}\rangle = Q_Y \,\mathbf{in}\, P;\ C$. We provide the derivation for this process:

$$\dfrac{\Delta, X:\tilde{U};\tilde{x}:\tilde{U}\vdash Q_X;\ \tilde{C} \qquad \dfrac{\Delta, X:\tilde{U}, Y:\tilde{U};\tilde{y}:\tilde{U}\vdash Q_Y;\tilde{C}' \qquad \Delta, X:\tilde{U}, Y:\tilde{U};\Gamma\vdash P;\ C}{\Delta, X:\tilde{U};\Gamma\vdash \mathbf{def}\, Y\langle\tilde{y}:\tilde{U}\rangle = Q_Y \,\mathbf{in}\, P;\ C}\,[\text{TDEF}]}{\Delta;\ \Gamma\vdash \mathbf{def}\, X\langle\tilde{x}:\tilde{U}\rangle = Q_X \,\mathbf{in}\, \mathbf{def}\, Y\langle\tilde{y}:\tilde{U}\rangle = Q_Y \,\mathbf{in}\, P;\ C}\,[\text{TDEF}]$$

By Proposition 4.5.2 $\Delta, X:\tilde{U}, Y:\tilde{U};\tilde{y}:\tilde{U}\vdash Q_Y;\tilde{C}'$ we obtain: $\Delta, Y:\tilde{U};\tilde{y}:\tilde{U}\vdash Q_Y;\tilde{C}'$.

By rearranging the premises:

$$\dfrac{\dfrac{\Delta, X:\tilde{U};\tilde{x}:\tilde{U}\vdash Q_X;\ \tilde{C} \qquad \Delta, X:\tilde{U}, Y:\tilde{U};\Gamma\vdash P;\ C}{\Delta, Y:\tilde{U};\Gamma\vdash \mathbf{def}\, X\langle\tilde{x}:\tilde{U}\rangle = Q_Y \,\mathbf{in}\, P;\ C}\,[\text{TDEF}] \qquad \Delta, Y:\tilde{U};\tilde{y}:\tilde{U}\vdash Q_Y;\tilde{C}'}{\Delta;\ \Gamma\vdash \mathbf{def}\, Y\langle\tilde{y}:\tilde{U}\rangle = Q_Y \,\mathbf{in}\, \mathbf{def}\, X\langle\tilde{x}:\tilde{U}\rangle = Q_X \,\mathbf{in}\, P;\ C}\,[\text{TDEF}]$$

Right to left: From the induction hypothesis: $\Delta;\ \Gamma\vdash \mathbf{def}\, Y\langle\tilde{y}:\tilde{U}\rangle = Q_Y \,\mathbf{in}\, \mathbf{def}\, X\langle\tilde{x}:\tilde{U}\rangle = Q_X \,\mathbf{in}\, P;\ C$. We provide the derivation for this process:

$$\dfrac{\dfrac{\dfrac{\Delta, Y:\tilde{U}, X:\tilde{U}; \tilde{x}:\tilde{U} \vdash Q_X;\ \tilde{C} \qquad \Delta, X:\tilde{U}, Y:\tilde{U}; \Gamma \vdash P;\ C}{\Delta, Y:\tilde{U}; \Gamma \vdash \mathbf{def}\, X\langle \tilde{x}:\tilde{U}\rangle = Q_Y \,\mathbf{in}\, P;\ C} \quad \Delta, Y:\tilde{U}; \tilde{y}:\tilde{U} \vdash Q_Y; \tilde{C}'}{\Delta;\ \Gamma \vdash \mathbf{def}\, Y\langle \tilde{y}:\tilde{U}\rangle = Q_Y \,\mathbf{in}\, \mathbf{def}\, X\langle \tilde{x}:\tilde{U}\rangle = Q_X \,\mathbf{in}\, P;\ C}\ [\text{TDEF}]}{}$$

By Proposition 4.5.2 $\Delta, Y:\tilde{U}, X:\tilde{U}; \tilde{x}:\tilde{U} \vdash Q_X;\ \tilde{C}$ we obtain: $\Delta, X:\tilde{U}; \tilde{x}:\tilde{U} \vdash Q_X;\ \tilde{C}$.

By rearranging the premises:

$$\dfrac{\Delta, X:\tilde{U}; \tilde{x}:\tilde{U} \vdash Q_X;\ \tilde{C} \qquad \dfrac{\Delta, Y:\tilde{U}; \tilde{y}:\tilde{U} \vdash Q_Y; \tilde{C}' \qquad \Delta, X:\tilde{U}, Y:\tilde{U}; \Gamma \vdash P;\ C}{\Delta, X:\tilde{U}; \Gamma \vdash \mathbf{def}\, Y\langle \tilde{y}:\tilde{U}\rangle = Q_Y \,\mathbf{in}\, P;\ C}\ [\text{TDEF}]}{\Delta;\ \Gamma \vdash \mathbf{def}\, X\langle \tilde{x}:\tilde{U}\rangle = Q_X \,\mathbf{in}\, \mathbf{def}\, Y\langle \tilde{y}:\tilde{U}\rangle = Q_Y \,\mathbf{in}\, P;\ C}\ [\text{TDEF}]$$

□

Lemma 4.5.1. By induction on the derivation of $\Delta;\ \Gamma, x:U \vdash P;\ C$ and $\Delta;\ \Gamma, \rho:\{\rho \mapsto S\} \vdash P;\ C \otimes \{\rho \mapsto S\}$, with a case analysis on the last rule applied.

- **0**

  1. We have $\Delta;\ \Gamma, \mathtt{x}:U \vdash \mathbf{0};\ C$ and $\Gamma' \vdash \mathtt{v}:U; \varnothing$

     $$\dfrac{C \text{ terminated}}{\Delta;\ \Gamma, \Gamma' \vdash \mathbf{0}\{v/x\};\ C}\ [\text{TINACT}]$$

  2. We have $\Delta;\ \Gamma, \rho:\{\rho \mapsto S\} \vdash \mathbf{0};\ C \otimes \{\rho \mapsto S\}$ and $\Gamma' \vdash \rho':\{\rho' \mapsto S\}; \{\rho' \mapsto S\}$ and $(\Gamma, \Gamma';\ C \otimes \{\rho' \mapsto S\})$ consistent

     $$\dfrac{C \otimes \{\rho' \mapsto S\} \text{ terminated}}{\Delta;\ \Gamma, \Gamma' \vdash \mathbf{0}\{\rho'/\rho\};\ C \otimes \{\rho' \mapsto S\}}\ [\text{TINACT}]$$

- $P \mid Q$

  1. $\Delta;\ \Gamma, \mathtt{x}:U \vdash (P \mid Q);\ C, \Gamma' \vdash \mathtt{v}:U; \varnothing$ From inversion on [TPAR] we have $\Delta;\ \Gamma, \mathtt{x}:U \vdash P;\ C_1$ and $\Delta;\ \Gamma, \mathtt{x}:U \vdash Q;\ C_2$.

     By applying the induction hypothesis on the two premises we obtain: $\Delta;\ \Gamma, \Gamma' \vdash P\{v/x\};\ C_1$ and $\Delta;\ \Gamma, \Gamma' \vdash Q\{v/x\};\ C_2$. Therefore, we conclude by:

     $$\dfrac{\Delta;\ \Gamma, \Gamma' \vdash P\{v/x\};\ C_1 \qquad \Delta;\ \Gamma, \Gamma' \vdash Q\{v/x\};\ C_2}{\Delta;\ \Gamma, \Gamma' \vdash (P \mid Q)\{v/x\};\ C_1 \otimes C_2}\ [\text{TPAR}]$$

2. $\Delta;\Gamma,\rho:\{\rho\mapsto S\}\vdash(P\,|\,Q);C\otimes\{\rho\mapsto S\},\Gamma'\vdash\rho':\{\rho'\mapsto S\};\{\rho'\mapsto S\}$ and $(\Gamma,\Gamma';C\otimes\{\rho'\mapsto S\})$ consistent

   If $\{\rho\mapsto S\}$ belongs to process P then:

$$\frac{\Delta;\ \Gamma,\rho:\{\rho\mapsto S\}\vdash P;\ C_1\otimes\{\rho\mapsto S\}\qquad\Delta;\ \Gamma,\rho:\{\rho\mapsto S\}\vdash Q;\ C_2}{\Delta;\ \Gamma,\rho:\{\rho\mapsto S\}\vdash P\,|\,Q;\ C\otimes\{\rho\mapsto S\}}\ [\text{TPAR}]$$

   By applying the induction hypothesis on the first premise we obtain: $\Delta;\Gamma,\Gamma'\vdash P\{\rho'/\rho\};\ C_1\otimes\{\rho'\mapsto S\}$. By applying Proposition 4.5.2 to the second premise we obtain: $\Delta;\ \Gamma\vdash Q;\ C_2$, and then by Proposition 4.5.1 $\Delta;\ \Gamma,\Gamma'\vdash Q;\ C_2$. Therefore:

$$\frac{\Delta;\ \Gamma,\Gamma'\vdash P\{\rho'/\rho\};\ C_1\otimes\{\rho'\mapsto S\}\qquad\Delta;\ \Gamma,\Gamma'\vdash Q;\ C_2}{\Delta;\ \Gamma,\Gamma'\vdash(P\,|\,Q)\{\rho'/\rho\};\ C\otimes\{\rho'\mapsto S\}}\ [\text{TPAR}]$$

   Or, if $\{\rho\mapsto S\}$ belongs to process Q then:

$$\frac{\Delta;\ \Gamma,\{\rho\mapsto S\}\vdash P;\ C_1\qquad\Delta;\ \Gamma,\{\rho\mapsto S\}\vdash Q\{\rho'/\rho\};\ C_2\otimes\{\rho'\mapsto S\}}{\Delta;\ \Gamma,\{\rho\mapsto S\}\vdash(P\,|\,Q)\{\rho'/\rho\};\ C\otimes\{\rho'\mapsto S\}}\ [\text{TPAR}]$$

   By applying the induction hypothesis on the second premise we obtain: $\Delta;\ \Gamma,\Gamma'\vdash Q\{\rho'/\rho\};\ C_2\otimes\{\rho'\mapsto S\}$. By applying Proposition 4.5.2 to the first premise we obtain: $\Delta;\ \Gamma\vdash P;\ C_1$, and then by Proposition 4.5.1 $\Delta;\ \Gamma,\Gamma'\vdash P;\ C_1$. Therefore:

$$\frac{\Delta;\ \Gamma,\Gamma'\vdash P;\ C_1\qquad\Delta;\ \Gamma,\Gamma'\vdash Q\{\rho'/\rho\};\ C_2\otimes\{\rho'\mapsto S\}}{\Delta;\ \Gamma,\Gamma'\vdash(P\,|\,Q)\{\rho'/\rho\};\ C\otimes\{\rho'\mapsto S\}}\ [\text{TPAR}]$$

- $(\nu\, s)P$

  1. $\Delta;\ \Gamma,\mathtt{x}:U\vdash(\nu\, s)P;\ C,\Gamma''\vdash\mathtt{v}:U;\varnothing$

$$\frac{\Delta;\ \Gamma,\mathtt{x}:U,\Gamma'\vdash P;\ C\otimes C'}{\Delta;\ \Gamma,\mathtt{x}:U\vdash(\nu\, s:\Gamma')P;\ C}\ [\text{TRES}]$$

     By applying the induction hypothesis on the premise we obtain: $\Delta;\ \Gamma,\Gamma',\Gamma''\vdash P\{v/x\};\ C\otimes C'$. Therefore:

$$\frac{\Delta;\ \Gamma,\Gamma',\Gamma''\vdash P\{v/x\};\ C\otimes C'}{\Delta;\ \Gamma,\Gamma'\vdash(\nu\, s:\Gamma')P\{v/x\};\ C}\ [\text{TRES}]$$

  2. $\Delta;\ \Gamma,\rho:\{\rho\mapsto S\}\vdash(\nu\, s)P;\ C\otimes\{\rho\mapsto S\},\ \Gamma''\vdash\rho':\{\rho'\mapsto S\};\{\rho'\mapsto S\}$ and $(\Gamma,\Gamma';\ C\otimes$

$\{\rho' \mapsto S\}$) consistent

$$\frac{\Delta;\ \Gamma, \rho : \{\rho \mapsto S\}, \Gamma' \vdash P\{\rho'/\rho\};\ C \otimes \{\rho' \mapsto S\} \otimes C'}{\Delta;\ \Gamma \vdash (\nu\, s : \Gamma')P\{\rho'/\rho\};\ C \otimes \{\rho' \mapsto S\}}\ [\text{TRES}]$$

By applying the induction hypothesis on the premise we obtain: $\Delta;\ \Gamma, \Gamma', \Gamma'' \vdash P\{\rho'/\rho\};\ C \otimes \{\rho' \mapsto S\} \otimes C'$. Therefore:

$$\frac{\Delta;\ \Gamma, \Gamma', \Gamma'' \vdash P\{\rho'/\rho\};\ C \otimes \{\rho' \mapsto S\} \otimes C'}{\Delta;\ \Gamma, \Gamma' \vdash (\nu\, s : \Gamma')P\{\rho'/\rho\};\ C \otimes \{\rho' \mapsto S\}}\ [\text{TRES}]$$

- $\mathtt{c}[\mathbf{p}]\&_{i \in I}\{l_i(x_i).P_i\}$

  1. $\Delta;\ \Gamma, \mathtt{w} : U \vdash \mathtt{c}[\mathbf{p}]\&_{i \in I}\{l_i(x_i).P_i\};\ C,\ \Gamma' \vdash \mathtt{v} : U; \varnothing$. There are several sub-cases, depending on the position of w and the form of U. If w is not involved in the input:

$$\frac{\Delta;\ \Gamma, \mathtt{w} : U, x_i : U_i \vdash P_i;\ C \otimes C_i \otimes \{\rho \mapsto S_i\} \quad \mathtt{c} : \mathtt{tr}(\rho), \rho : \{\rho \mapsto S_i\} \in \Gamma \quad \forall i \in I}{\Delta;\ \Gamma, \mathtt{w} : U \vdash \mathtt{c}[\mathbf{p}]\&_{i \in I}\{l_i(x_i).P_i\};\ C \otimes \{\rho \mapsto \mathbf{p}\ \&_{i \in I}?l_i(U_i).S_i\}}\ [\text{TBR}]$$

  By applying the induction hypothesis on the premise we obtain: $\Delta; \Gamma, \Gamma', x_i : U_i \vdash P\{v/w\}; C \otimes \{\rho' \mapsto S\} \otimes C'$. Therefore:

$$\frac{\Delta;\ \Gamma, \Gamma', x_i : U_i \vdash P\{v/w\};\ C \otimes \{\rho' \mapsto S\} \otimes C' \quad \mathtt{c} : \mathtt{tr}(\rho), \rho : \{\rho \mapsto S_i\} \in \Gamma \quad \forall i \in I}{\Delta;\ \Gamma, \Gamma' \vdash \mathtt{c}[\mathbf{p}]\&_{i \in I}\{l_i(x_i).P_i\}\{v/w\};\ C \otimes \{\rho \mapsto \mathbf{p}\ \&_{i \in I}?l_i(U_i).S_i\}}\ [\text{TBR}]$$

  If w is the input channel:

$$\frac{\Delta;\ \Gamma'', \mathtt{w} : \mathtt{tr}(\rho), \rho : \{\rho \mapsto S_i\}, x_i : U_i \vdash P_i;\ C \otimes C_i \otimes \{\rho \mapsto S_i\} \quad \forall i \in I}{\Delta; \Gamma'', \mathtt{w} : \mathtt{tr}(\rho), \rho : \{\rho \mapsto \mathbf{p}\ \&_{i \in I}?l_i(U_i).S_i\} \vdash \mathtt{w}[\mathbf{p}]\&_{i \in I}\{l_i(x_i).P_i\}; C \otimes \{\rho \mapsto \mathbf{p}\ \&_{i \in I}?l_i(U_i).S_i\}}\ [\text{TBR}]$$

  By applying the induction hypothesis on the premise we obtain: $\Delta;\ \Gamma'', \Gamma', \rho : \{\rho \mapsto S_i\}, x_i : U_i \vdash P_i\{v/w\};\ C \otimes C_i \otimes \{\rho \mapsto S_i\}$.

  Therefore:

$$\frac{\Delta;\ \Gamma'', \Gamma', \rho : \{\rho \mapsto S_i\}, x_i : U_i \vdash P_i\{v/w\};\ C \otimes C_i \otimes \{\rho \mapsto S_i\} \quad \forall i \in I}{\Delta; \Gamma'', \Gamma', \rho : \{\rho \mapsto \mathbf{p}\ \&_{i \in I}?l_i(U_i).S_i\} \vdash \mathtt{w}[\mathbf{p}]\&_{i \in I}\{l_i(x_i).P_i\}\{v/w\}; C \otimes \{\rho \mapsto \mathbf{p}\ \&_{i \in I}?l_i(U_i).S_i\}}\ [\text{TBR}]$$

  2. $\Delta;\ \Gamma, \rho : \{\rho \mapsto S\} \vdash \mathtt{c}[\mathbf{p}]\&_{i \in I}\{l_i(x_i).P_i\};\ C \otimes \{\rho \mapsto S\},\quad \Gamma' \vdash \rho' : \{\rho' \mapsto S\}; \{\rho' \mapsto S\}$ and

$(\Gamma,\Gamma'; C\otimes\{\rho'\mapsto S\})$ consistent

$$\frac{\Delta;\ \Gamma,\rho:\{\rho\mapsto S\},x_i:U_i\vdash P_i;\ C\otimes C_i\otimes\{\rho_c\mapsto S_i\}\otimes\{\rho\mapsto S\}\quad \mathtt{c}:\mathtt{tr}(\rho_c),\rho:\{\rho_c\mapsto S_i\}\in\Gamma\qquad\forall i\in I}{\Delta;\ \Gamma,\rho:\{\rho\mapsto S\}\vdash \mathtt{c}[\mathbf{p}]\&_{i\in I}\{l_i(x_i).P_i\};\ C\otimes\{\rho_c\mapsto \mathbf{p}\ \&_{i\in I}?l_i\,(U_i)\,.S_i\}\otimes\{\rho\mapsto S\}}\ [\text{TBR}]$$

By applying the induction hypothesis on the premise we obtain:
$\Delta;\ \Gamma,\Gamma',x_i:U_i,\mathtt{c}:\mathtt{tr}(\rho)\vdash P_i\{\rho'/\rho\};\ C\otimes C_i\otimes\{\rho_c\mapsto S_i\}\otimes\{\rho'\mapsto S\}$. Therefore:

$$\frac{\Delta;\ \Gamma,\Gamma',x_i:U_i\vdash P_i\{\rho'/\rho\};\ C\otimes C_i\otimes\{\rho_c\mapsto S_i\}\otimes\{\rho'\mapsto S\}\quad \mathtt{c}:\mathtt{tr}(\rho_c),\rho:\{\rho_c\mapsto S_i\}\in\Gamma\qquad\forall i\in I}{\Delta;\ \Gamma,\Gamma'\vdash \mathtt{c}[\mathbf{p}]\&_{i\in I}\{l_i(x_i).P_i\}\{\rho'/\rho\};\ C\otimes\{\rho_c\mapsto \mathbf{p}\ \&_{i\in I}?l_i\,(U_i)\,.S_i\}\otimes\{\rho'\mapsto S\}}\ [\text{TBR}]$$

- $\mathtt{c}[\mathbf{p}]\oplus\langle l(v)\rangle.P$

  1. $\Delta;\ \Gamma,\mathtt{x}:U\vdash \mathtt{c}[\mathbf{p}]\oplus\langle l(v)\rangle.P;\ C$, and $\Gamma'\vdash \mathtt{w}:U;\varnothing$

     There are several sub-cases, depending on the position of x and the form of U. If x is not involved in the output:

     $$\frac{\Gamma,\mathtt{x}:U\vdash v:U;\ C\quad \Delta;\ \Gamma,\mathtt{x}:U\vdash P;C'\otimes\{\rho\mapsto S_j\}\quad \mathtt{c}:\mathtt{tr}(\rho)\in\Gamma\quad j\in I}{\Delta;\ \Gamma,\mathtt{x}:U\vdash \mathtt{c}[\mathbf{p}]\oplus\langle l_j(v)\rangle.P;\ C\otimes C'\otimes\{\rho\mapsto \mathbf{p}\ \oplus_{i\in I}!l_i(U_i).S_i\}}\ [\text{TSEL}]$$

     By applying the induction hypothesis on the premise we obtain:
     $\Delta;\ \Gamma,\Gamma'\vdash P\{w/x\};C'\otimes\{\rho\mapsto S_j\}$. Therefore:

     $$\frac{\Gamma,\Gamma'\vdash v:U;\ C\quad \Delta;\ \Gamma,\Gamma'\vdash P\{w/x\};C'\otimes\{\rho\mapsto S_j\}\quad \mathtt{c}:\mathtt{tr}(\rho)\in\Gamma\quad j\in I}{\Delta;\ \Gamma,\Gamma'\vdash \mathtt{c}[\mathbf{p}]\oplus\langle l_j(v)\rangle.P\{w/x\};\ C\otimes C'\otimes\{\rho\mapsto \mathbf{p}\ \oplus_{i\in I}!l_i(U_i).S_i\}}\ [\text{TSEL}]$$

     If x is the output name:

     $$\frac{\Gamma,\mathtt{x}:U\vdash x:U;\ C\quad \Delta;\ \Gamma,\mathtt{x}:U\vdash P;C'\otimes\{\rho\mapsto S_j\}\quad \mathtt{c}:\mathtt{tr}(\rho)\in\Gamma,\mathtt{x}:U\quad j\in I}{\Delta;\ \Gamma,\mathtt{x}:U\vdash \mathtt{c}[\mathbf{p}]\oplus\langle l_j(v)\rangle.P;\ C\otimes C'\otimes\{\rho\mapsto \mathbf{p}\ \oplus_{i\in I}!l_i(U_i).S_i\}}\ [\text{TSEL}]$$

     By applying the induction hypothesis on the premises we obtain:
     $\Gamma,\Gamma'\vdash w:U;\ C$ and $\Delta;\ \Gamma,\Gamma'\vdash P\{w/x\};C'\otimes\{\rho\mapsto S_j\}$. Therefore:

     $$\frac{\Gamma,\Gamma'\vdash w:U;\ C\quad \Delta;\ \Gamma,\Gamma'\vdash P\{w/x\};C'\otimes\{\rho\mapsto S_j\}\quad \mathtt{c}:\mathtt{tr}(\rho)\in\Gamma\quad j\in I}{\Delta;\ \Gamma,\Gamma'\vdash \mathtt{c}[\mathbf{p}]\oplus\langle l_j(v)\rangle.P\{w/x\};\ C\otimes C'\otimes\{\rho\mapsto \mathbf{p}\ \oplus_{i\in I}!l_i(U_i).S_i\}}\ [\text{TSEL}]$$

     If x is the output channel:

$$\frac{\Gamma \vdash v : U;\ C \quad \Delta;\ \Gamma \vdash P; C' \otimes \{\rho \mapsto S_j\} \quad \mathtt{c} : \mathtt{tr}(\rho) \in \Gamma \quad j \in I}{\Delta;\ \Gamma \vdash \mathtt{c}[\mathbf{p}] \oplus \langle l_j(v)\rangle.P;\ C \otimes C' \otimes \{\rho \mapsto \mathbf{p} \oplus_{i \in I} !l_i(U_i).S_i\}}\ [\text{TSEL}]$$

2. $\Delta;\ \Gamma, \rho : \{\rho \mapsto S\} \vdash \mathtt{c}[\mathbf{p}] \oplus \langle l(v)\rangle.P;\ C \otimes \{\rho \mapsto S\}, \Gamma' \vdash \rho' : \{\rho' \mapsto S\}; \{\rho' \mapsto S\}$ and $(\Gamma, \Gamma';\ C \otimes \{\rho' \mapsto S\})$ consistent

$$\frac{\Gamma \vdash v : U;\ C \quad \Delta;\ \Gamma \vdash P; C' \otimes \{\rho \mapsto S_j\} \quad \mathtt{c} : \mathtt{tr}(\rho) \in \Gamma \quad j \in I}{\Delta;\ \Gamma \vdash \mathtt{c}[\mathbf{p}] \oplus \langle l_j(v)\rangle.P;\ C \otimes C' \otimes \{\rho \mapsto \mathbf{p} \oplus_{i \in I} !l_i(U_i).S_i\}}\ [\text{TSEL}]$$

- $\mathtt{c}[\mathbf{p}] \oplus \langle l(\mathsf{pack}(\rho, \mathtt{c}'))\rangle.P$

1. $\Delta;\ \Gamma, \mathtt{x} : U \vdash \mathtt{c}[\mathbf{p}] \oplus \langle l(\mathsf{pack}(\rho, \mathtt{c}'))\rangle.P;\ C$, and $\Gamma' \vdash \mathtt{v} : U; \varnothing$

[TSELP]
$$\frac{\Gamma \vdash v : \mathtt{tr}(\rho'); \varnothing \qquad \Delta;\ \Gamma \vdash P;\ C \otimes \{\rho \mapsto S_j\} \otimes \{\rho' \mapsto U\} \qquad \mathtt{c} : \mathtt{tr}(\rho) \in \Gamma \qquad j \in I}{\Delta; \Gamma \vdash \mathtt{c}[\mathbf{p}] \oplus \langle l_j(\mathsf{pack}(\rho', v))\rangle.P; C \otimes \{\rho \mapsto \mathbf{p} \oplus_{j \in I} !l_j(\exists[\rho' | \{\rho' \mapsto U\}].\mathtt{tr}(\rho')).S\} \otimes \{\rho' \mapsto U\}}$$

2. $\Delta;\ \Gamma, \rho : \{\rho \mapsto S\} \vdash \mathtt{c}[\mathbf{p}] \oplus \langle l(\mathsf{pack}(\rho, \mathtt{c}'))\rangle.P;\ C \otimes \{\rho \mapsto S\}$,
$\Gamma' \vdash \rho' : \{\rho' \mapsto S\}; \{\rho' \mapsto S\}$ and $(\Gamma, \Gamma';\ C \otimes \{\rho' \mapsto S\})$ consistent

[TSELP]
$$\frac{\Gamma \vdash v : \mathtt{tr}(\rho'); \varnothing \qquad \Delta;\ \Gamma \vdash P;\ C \otimes \{\rho \mapsto S_j\} \otimes \{\rho' \mapsto U\} \qquad \mathtt{c} : \mathtt{tr}(\rho) \in \Gamma \qquad j \in I}{\Delta; \Gamma \vdash \mathtt{c}[\mathbf{p}] \oplus \langle l_j(\mathsf{pack}(\rho', v))\rangle.P; C \otimes \{\rho \mapsto \mathbf{p} \oplus_{j \in I} !l_j(\exists[\rho' | \{\rho' \mapsto U\}].\mathtt{tr}(\rho')).S\} \otimes \{\rho' \mapsto U\}}$$

- $\mathtt{c}[\mathbf{p}] \& _{i \in I} \{l_i(\mathsf{pack}(\rho, \mathtt{c}')_i).P_i\}$

1. $\Delta;\ \Gamma, \mathtt{x} : U \vdash \mathtt{c}[\mathbf{p}] \& _{i \in I} \{l_i(\mathsf{pack}(\rho, \mathtt{c}')_i).P_i\};\ C$, and $\Gamma' \vdash \mathtt{v} : U; \varnothing$

[TBRP]
$$\frac{\Delta;\ \Gamma, \mathtt{v_i} : \mathtt{tr}(\rho_i) \vdash P_i;\ C \otimes \{\rho \mapsto S_i\} \quad \forall i \in I \quad \mathtt{c} : \mathtt{tr}(\rho) \in \Gamma}{\Delta;\ \Gamma \vdash \mathtt{c}[\mathbf{p}] \& _{i \in I} \{l_i(\mathsf{pack}(\rho_i, \mathtt{v_i})).P_i\};\ C \otimes \{\rho \mapsto \mathbf{p}\ \&_{i \in I} ?l(\exists[\rho_i | \{\rho_i \mapsto U_i\}].\mathtt{tr}(\rho_i)).S_i\}}$$

2. $\Delta;\ \Gamma, \rho : \{\rho \mapsto S\} \vdash \mathtt{c}[\mathbf{p}] \& _{i \in I} \{l_i(\mathsf{pack}(\rho, \mathtt{c}')_i).P_i\}; C \otimes \{\rho \mapsto S\}$,
$\Gamma' \vdash \rho' : \{\rho' \mapsto S\}; \{\rho' \mapsto S\}$ and $(\Gamma, \Gamma';\ C \otimes \{\rho' \mapsto S\})$ consistent

[TBRP]
$$\frac{\Delta;\ \Gamma, \mathtt{v_i} : \mathtt{tr}(\rho_i) \vdash P_i;\ C \otimes \{\rho \mapsto S_i\} \quad \forall i \in I \quad \mathtt{c} : \mathtt{tr}(\rho) \in \Gamma}{\Delta;\ \Gamma \vdash \mathtt{c}[\mathbf{p}] \& _{i \in I} \{l_i(\mathsf{pack}(\rho_i, \mathtt{v_i})).P_i\}; C \otimes \{\rho \mapsto \mathbf{p}\ \&_{i \in I} ?l(\exists[\rho_i | \{\rho_i \mapsto U_i\}].\mathtt{tr}(\rho_i)).S_i\}}$$

- **def** D **in** P

1. $\Delta;\ \Gamma, \mathtt{x} : U \vdash P;\ C, \Gamma' \vdash \mathtt{v} : U; \varnothing$

2. $\Delta;\ \Gamma, \rho : \{\rho \mapsto S\} \vdash P;\ C \otimes \{\rho \mapsto S\}$,
$\Gamma' \vdash \rho' : \{\rho' \mapsto S\}; \{\rho' \mapsto S\}$ and $(\Gamma, \Gamma';\ C \otimes \{\rho' \mapsto S\})$ consistent

$$\text{[TDEF]}\quad \frac{\Delta, X : \tilde{U}; \tilde{x} : \tilde{U} \vdash P;\ \tilde{C} \qquad \Delta, X : \tilde{U}; \Gamma \vdash Q;\ C}{\Delta;\ \Gamma \vdash \mathbf{def}\ X\langle \tilde{x} : \tilde{U} \rangle = P\ \mathbf{in}\ Q;\ C}$$

- $X\langle \tilde{x} \rangle$

  1. $\Delta;\ \Gamma, \mathrm{x} : U \vdash P;\ C, \Gamma' \vdash \mathrm{v} : U; \varnothing$
  2. $\Delta;\ \Gamma, \rho : \{\rho \mapsto S\} \vdash P;\ C \otimes \{\rho \mapsto S\}, \Gamma' \vdash \rho' : \{\rho' \mapsto S\}; \{\rho' \mapsto S\}$ and $(\Gamma, \Gamma';\ C \otimes \{\rho' \mapsto S\})$ consistent

$$\text{[TCALL]}\quad \frac{\forall i \in \{1..n\} \qquad \Gamma \vdash v_i : U_i;\ C_i}{\Delta, x : U_1, \ldots, U_n; \Gamma \vdash X\langle v_1, \ldots, v_n \rangle;\ C_1 \otimes \ldots \otimes C_n}$$

□

Bibliography

[1] Jonathan Aldrich, Joshua Sunshine, Darpan Saini, and Zachary Sparks. Typestate-oriented programming. In *OOPSLA '09*, pages 1015–1022. ACM Press, 2009.

[2] Chris Allan, Pavel Avgustinov, Aske Simon Christensen, Laurie J. Hendren, Sascha Kuzins, Ondrej Lhoták, Oege de Moor, Damien Sereni, Ganesh Sittampalam, and Julian Tibble. Adding trace matching with free variables to AspectJ. In *OOPSLA*, pages 345–364, 2005.

[3] Antlr project homepage. www.antlr.org.

[4] Robert Atkey. Parameterised notions of computation. *J. Funct. Program.*, 19(3-4):335–376, 2009.

[5] Pedro Baltazar, Dimitris Mostrous, and Vasco Thudichum Vasconcelos. Linearly refined session types. In *LINEARITY*, volume 101 of *EPTCS*, pages 38–49, 2012.

[6] Stephanie Balzer and Frank Pfenning. Manifest sharing with session types. *PACMPL*, 1(ICFP):37:1–37:29, 2017.

[7] Stephanie Balzer, Bernardo Toninho, and Frank Pfenning. Manifest deadlock-freedom for shared session types. In *ESOP*, volume 11423 of *LNCS*, pages 611–639. Springer, 2019.

[8] Nels E. Beckman, Kevin Bierhoff, and Jonathan Aldrich. Verifying correct usage of atomic blocks and typestate. In *OOPSLA '08*, pages 227–244. ACM Press, 2008.

[9] Giovanni Bernardi and Matthew Hennessy. Using higher-order contracts to model session types (extended abstract). In *CONCUR*, volume 8704 of *Lecture Notes in Computer Science*, pages 387–401. Springer, 2014.

[10] Jean-Philippe Bernardy, Mathieu Boespflug, Ryan R. Newton, Simon Peyton Jones, and Arnaud Spiwack. Linear haskell: practical linearity in a higher-order polymorphic language. *PACMPL*, 2(POPL):5:1–5:29, 2018.

[11] Lorenzo Bettini, Mario Coppo, Loris D'Antoni, Marco De Luca, Mariangiola Dezani-Ciancaglini, and Nobuko Yoshida. Global progress in dynamically interleaved multiparty sessions. In Franck van Breugel and Marsha Chechik, editors, *CONCUR 2008 - Concurrency Theory*, pages 418–433, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[12] Kevin Bierhoff and Jonathan Aldrich. Modular typestate checking of aliased objects. In *OOPSLA '07*, pages 301–320. ACM Press, 2007.

[13] Kevin Bierhoff and Jonathan Aldrich. PLURAL: checking protocol compliance under aliasing. In *ICSE Companion*, pages 971–972. ACM Press, 2008.

[14] Kevin Bierhoff, Nels E. Beckman, and Jonathan Aldrich. Practical api protocol checking with access permissions. In *ECOOP '09*, volume 5653 of *Springer LNCS*, pages 195–219, 2009.

[15] Laura Bocchi, Tzu-Chun Chen, Romain Demangeon, Kohei Honda, and Nobuko Yoshida. Monitoring networks through multiparty session types. In *FMOODS/FORTE*, volume 7892 of *Lecture Notes in Computer Science*, pages 50–65. Springer, 2013.

[16] Eric Bodden and Laurie J. Hendren. The clara framework for hybrid typestate analysis. *Software Tools for Technology Transfer*, 14(3):307–326, 2012.

[17] Viviana Bono and Luca Padovani. Typing copyless message passing. *Logical Methods in Computer Science*, 8(1), 2012.

[18] Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, 1983.

[19] Luís Caires and Jorge A. Pérez. Linearity, control effects, and behavioral types. In *ESOP*, volume 10201 of *LNCS*, pages 229–259. Springer, 2017.

[20] Luís Caires and Frank Pfenning. Session types as intuitionistic linear propositions. In *CONCUR*, volume 6269 of *LNCS*, pages 222–236. Springer, 2010.

[21] Sara Capecchi, Mario Coppo, Mariangiola Dezani-Ciancaglini, Sophia Drossopoulou, and Elena Giachino. Amalgamating sessions and methods in object-oriented languages with generics. *Theoret. Comp. Sci.*, 410:142–167, 2009.

[22] Marco Carbone, Kohei Honda, and Nobuko Yoshida. Structured communication-centered programming for web services. *ACM Trans. Program. Lang. Syst.*, 34(2):8:1–8:78, 2012.

[23] Marco Carbone, Sam Lindley, Fabrizio Montesi, Carsten Schürmann, and Philip Wadler. Coherence generalises duality: A logical explanation of multiparty session types. In *CONCUR*, volume 59 of *LIPIcs*, pages 33:1–33:15. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2016.

[24] Marco Carbone, Fabrizio Montesi, Carsten Schürmann, and Nobuko Yoshida. Multiparty session types as coherence proofs. In *CONCUR*, volume 42 of *LIPIcs*. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2015.

[25] Giuseppe Castagna, Rocco De Nicola, and Daniele Varacca. Semantic subtyping for the pi-calculus. *Theor. Comput. Sci.*, 398(1-3):217–242, 2008.

[26] Giuseppe Castagna, Mariangiola Dezani-Ciancaglini, Elena Giachino, and Luca Padovani. Foundations of session types. In *PPDP*, pages 219–230. ACM, 2009.

[27] Giuseppe Castagna, Nils Gesbert, and Luca Padovani. A theory of contracts for web services. In *POPL*, pages 261–272. ACM, 2008.

[28] Tushar D Chandra, Robert Griesemer, and Joshua Redstone. Paxos made live: an engineering perspective. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, pages 398–407. ACM, 2007.

[29] Tzu-Chun Chen, Laura Bocchi, Pierre-Malo Deniélou, Kohei Honda, and Nobuko Yoshida. Asynchronous distributed monitoring for multiparty session enforcement. In *TGC*, volume 7173 of *Lecture Notes in Computer Science*, pages 25–45. Springer, 2011.

[30] Tzu-Chun Chen, Mariangiola Dezani-Ciancaglini, Alceste Scalas, and Nobuko Yoshida. On the preciseness of subtyping in session types. *CoRR*, abs/1610.00328, 2016.

[31] Tzu-Chun Chen, Mariangiola Dezani-Ciancaglini, Alceste Scalas, and Nobuko Yoshida. On the preciseness of subtyping in session types. *Logical Methods in Computer Science*, 13(2), 2017.

[32] Tzu-Chun Chen, Mariangiola Dezani-Ciancaglini, and Nobuko Yoshida. On the preciseness of subtyping in session types. In *PPDP*, pages 135–146. ACM, 2014.

[33] Ezra Cooper, Sam Lindley, Philip Wadler, and Jeremy Yallop. Links: Web programming without tiers. In *FMCO*, volume 4709 of *Lecture Notes in Computer Science*, pages 266–296. Springer, 2006.

[34] Mario Coppo, Mariangiola Dezani-Ciancaglini, Luca Padovani, and Nobuko Yoshida. A gentle introduction to multiparty asynchronous session types. In *SFM*, volume 9104 of *Lecture Notes in Computer Science*, pages 146–178. Springer, 2015.

[35] Mario Coppo, Mariangiola Dezani-Ciancaglini, Nobuko Yoshida, and Luca Padovani. Global progress for dynamically interleaved multiparty sessions. *Mathematical Structures in Computer Science*, 26(2):238–302, 2016.

[36] Silvia Crafa and Luca Padovani. The chemical approach to typestate-oriented programming. *ACM Transactions on Programming Languages and Systems*, 39(3):13:1–13:45, 2017.

[37] Karl Crary, David Walker, and Greg Morrisett. Typed memory management in a calculus of capabilities. In *POPL*, pages 262–275. ACM, 1999.

[38] Ornela Dardha, Simon J Gay, Dimitrios Kouzapas, Roly Perera, A Laura Voinea, and Florian Weber. Mungo and StMungo: Tools for Typechecking Protocols in Java. In Simon J Gay and António Ravara, editors, *Behavioural Types: from Theory to Tools*, chapter 14, pages 309–328. River Publishers, 2017.

[39] Ornela Dardha, Elena Giachino, and Davide Sangiorgi. Session types revisited. In *PPDP*. ACM, 2012.

[40] Ornela Dardha, Elena Giachino, and Davide Sangiorgi. Session types revisited. *Inf. Comput.*, 256:253–286, 2017.

[41] Robert DeLine and Manuel Fähndrich. Enforcing high-level protocols in low-level software. In *PLDI*, pages 59–69. ACM Press, 2001.

[42] Robert DeLine and Manuel Fähndrich. Typestates for objects. In *ECOOP '04*, volume 3086 of *Springer LNCS*, pages 465–490, 2004.

[43] Romain Demangeon and Kohei Honda. Full abstraction in a subtyped pi-calculus with linear types. In *CONCUR*, volume 6901 of *Lecture Notes in Computer Science*, pages 280–296. Springer, 2011.

[44] Romain Demangeon and Kohei Honda. Nested protocols in session types. In *CONCUR*, volume 7454 of *Lecture Notes in Computer Science*, pages 272–286. Springer, 2012.

[45] Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty session types meet communicating automata. In *ESOP*, volume 7211 of *Lecture Notes in Computer Science*, pages 194–213. Springer, 2012.

[46] Pierre-Malo Deniélou, Nobuko Yoshida, Andi Bejleri, and Raymond Hu. Parameterised multiparty session types. *Logical Methods in Computer Science*, 8(4), 2012.

[47] Mariangiola Dezani-Ciancaglini, Sophia Drossopoulou, Elena Giachino, and Nobuko Yoshida. Bounded session types for object-oriented languages. *FMCO*, 4709:207–245, 2007.

[48] Mariangiola Dezani-Ciancaglini, Sophia Drossopoulou, Dimitris Mostrous, and Nobuko Yoshida. Objects and session types. *Information and Computation*, 207(5):595–641, 2009.

[49] Mariangiola Dezani-Ciancaglini, Silvia Ghilezan, Svetlana Jaksic, Jovanka Pantovic, and Nobuko Yoshida. Precise subtyping for synchronous multiparty sessions. In *PLACES*, pages 29–43, 2015.

[50] Mariangiola Dezani-Ciancaglini, Nobuko Yoshida, Alexander Ahern, and Sophia Drossopolou. A distributed object-oriented language with session types. In *TGC '05*, volume 3705 of *Springer LNCS*, pages 299–318, 2005.

[51] Manuel Fähndrich, Mark Aiken, Chris Hawblitzel, Orion Hodson, Galen Hunt, James R. Larus, and Steven Levi. Language support for fast and reliable message-based communication in Singularity OS. In *EuroSys*, pages 177–190. ACM Press, 2006.

[52] Manuel Fähndrich, Mark Aiken, Chris Hawblitzel, Orion Hodson, Galen C. Hunt, James R. Larus, and Steven Levi. Language support for fast and reliable message-based communication in singularity OS. In *EuroSys*, pages 177–190. ACM, 2006.

[53] Manuel Fähndrich and Robert DeLine. Adoption and focus: Practical linear types for imperative programming. In *PLDI*, pages 13–24. ACM, 2002.

[54] Roy T. Fielding and Julian F. Reschke. Hypertext transfer protocol (HTTP/1.1): message syntax and routing. *RFC*, 7230:1–89, 2014.

[55] Simon Fowler. An erlang implementation of multiparty session actors. In *ICE*, volume 223 of *EPTCS*, pages 36–50, 2016.

[56] Simon Fowler, Sam Lindley, J. Garrett Morris, and Sára Decova. Exceptional asynchronous session types: session types without tiers. *PACMPL*, 3(POPL):28:1–28:29, 2019.

[57] Juliana Franco and Vasco Thudichum Vasconcelos. A concurrent programming language with refined session types. In *SEFM Workshops*, volume 8368 of *Lecture Notes in Computer Science*, pages 15–28. Springer, 2013.

[58] Ronald Garcia, Éric Tanter, Roger Wolff, and Jonathan Aldrich. Foundations of typestate-oriented programming. *ACM Trans. Program. Lang. Syst.*, 36(4):12:1–12:44, 2014.

[59] Simon J. Gay. Bounded polymorphism in session types. *Mathematical Structures in Computer Science*, 18(5):895–930, 2008.

[60] Simon J. Gay. Subtyping supports safe session substitution. In *A List of Successes That Can Change the World*, volume 9600 of *Lecture Notes in Computer Science*, pages 95–108. Springer, 2016.

[61] Simon J. Gay and Malcolm J. Hole. Subtyping for session types in the pi calculus. *Acta Informatica*, 42(2/3):191–225, 2005.

[62] Simon J. Gay, Peter Thiemann, and Vasco T. Vasconcelos. Duality of session types: The final cut. In *PLACES@ETAPS*, volume 314 of *EPTCS*, pages 23–33, 2020.

[63] Simon J. Gay and Vasco T. Vasconcelos. Linear type theory for asynchronous session types. *Journal of Functional Programming*, 20(1):19–50, 2010.

[64] Simon J. Gay, Vasco T. Vasconcelos, António Ravara, Nils Gesbert, and Alexandre Z. Caldeira. Modular session types for distributed object-oriented programming. In *POPL '10*, pages 299–312. ACM Press, 2010.

[65] Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987.

[66] Jan Friso Groote, Aad Mathijssen, Michel Reniers, Yaroslav Usenko, and Muck Van Weerdenburg. The formal specification language mcrl2. In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2007.

[67] Görel Hedin. An introductory tutorial on jastadd attribute grammars. In *GTTSE*, volume 6491 of *Lecture Notes in Computer Science*, pages 166–200. Springer, 2009.

[68] Kohei Honda. Types for dyadic interaction. In *CONCUR*, volume 715 of *LNCS*, pages 509–523. Springer, 1993.

[69] Kohei Honda, Aybek Mukhamedov, Gary Brown, Tzu-Chun Chen, and Nobuko Yoshida. Scribbling interactions with a formal foundation. In *International Conference on Distributed Computing and Internet Technology*, pages 55–75. Springer, 2011.

[70] Kohei Honda, Aybek Mukhamedov, Gary Brown, Tzu-Chun Chen, and Nobuko Yoshida. Scribbling interactions with a formal foundation. In *ICDCIT 2011*, volume 6536 of *LNCS*. Springer, 2011.

[71] Kohei Honda, Vasco Thudichum Vasconcelos, and Makoto Kubo. Language primitives and type discipline for structured communication-based programming. In *ESOP*, volume 1381 of *Lecture Notes in Computer Science*, pages 122–138. Springer, 1998.

[72] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In *POPL*, pages 273–284. ACM, 2008.

[73] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. *J. ACM*, 63(1):9:1–9:67, 2016.

[74] Raymond Hu. Distributed Programming Using Java APIs Generated from Session Types. *Behavioural Types: from Theory to Tools*, pages 287–308, 2017.

[75] Raymond Hu, Dimitrios Kouzapas, Olivier Pernet, Nobuko Yoshida, and Kohei Honda. Type-safe eventful sessions in java. In *ECOOP '10*, volume 6183 of *Springer LNCS*, pages 329–353, 2010.

[76] Raymond Hu, Rumyana Neykova, Nobuko Yoshida, Romain Demangeon, and Kohei Honda. Practical interruptible conversations - distributed dynamic verification with session types and python. In *RV*, volume 8174 of *Lecture Notes in Computer Science*, pages 130–148. Springer, 2013.

[77] Raymond Hu and Nobuko Yoshida. Hybrid session verification through endpoint API generation. In *FASE 16*, volume 9633 of *Springer LNCS*, pages 401–418, 2016.

[78] Raymond Hu and Nobuko Yoshida. Explicit connection actions in multiparty session types. In *FASE*, volume 10202 of *Lecture Notes in Computer Science*, pages 116–133. Springer, 2017.

[79] Raymond Hu, Nobuko Yoshida, and Kohei Honda. Session-based distributed programming in Java. In *ECOOP '08*, volume 5142 of *Springer LNCS*, pages 516–541, 2008.

[80] Hans Hüttel et al. Foundations of session types and behavioural contracts. *ACM Computing Surveys*, 49(1), 2016.

[81] Atsushi Igarashi and Naoki Kobayashi. A generic type system for the pi-calculus. *Theoretical Computer Science*, 311(1-3):121–163, 2004.

[82] Keigo Imai, Nobuko Yoshida, and Shoji Yuen. Session-ocaml: A session-based library with polarities and lenses. In *COORDINATION*, volume 10319 of *Lecture Notes in Computer Science*, pages 99–118. Springer, 2017.

[83] Keigo Imai, Shoji Yuen, and Kiyoshi Agusa. Session type inference in haskell. In *PLACES*, volume 69 of *EPTCS*, pages 74–91, 2010.

[84] Naoki Kobayashi. Type systems for concurrent processes: From deadlock-freedom to livelock-freedom, time-boundedness. In *IFIP TCS*, volume 1872 of *Springer LNCS*, pages 365–389. Springer, 2000.

[85] Dimitrios Kouzapas, Ornela Dardha, Roly Perera, and Simon J. Gay. Typechecking protocols with Mungo and StMungo. In *PPDP*, pages 146–159. ACM, 2016.

[86] Dimitrios Kouzapas, Ornela Dardha, Roly Perera, and Simon J. Gay. Typechecking protocols with Mungo and StMungo: a session type toolchain for Java. *Science of Computer Programming*, 155:52–75, 2018.

[87] Dimitrios Kouzapas, Ramunas Forsberg Gutkovas, A Laura Voinea, and Simon J Gay. A Session Type System for Asynchronous Unreliable Broadcast Communication. *arXiv preprint arXiv:1902.01353*, 2019.

[88] Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, 1998.

[89] Leslie Lamport et al. Paxos made simple. *ACM Sigact News*, 32(4):18–25, 2001.

[90] Sam Lindley and J. Garrett Morris. Embedding session types in haskell. In *Haskell*, pages 133–145. ACM, 2016.

[91] Sam Lindley and J. Garrett Morris. Talking bananas: structural recursion for session types. In *ICFP*, pages 434–447. ACM, 2016.

[92] Sam Lindley and J Garrett Morris. Lightweight functional session types. *Behavioural Types: from Theory to Tools*, pages 265–286, 2017.

[93] Filipe Militão, Jonathan Aldrich, and Luís Caires. Aliasing control with view-based typestate. In *FTFJP*, pages 7:1–7:7. ACM, 2010.

[94] Dimitris Mostrous. *Session types in concurrent calculi : higher-order processes and objects*. PhD thesis, Imperial College London, UK, 2010.

[95] Dimitris Mostrous and Vasco T. Vasconcelos. Affine sessions. *Logical Methods in Computer Science*, 14(4), 2018.

[96] Dimitris Mostrous and Nobuko Yoshida. Session-based communication optimisation for higher-order mobile processes. In *TLCA*, volume 5608 of *Lecture Notes in Computer Science*, pages 203–218. Springer, 2009.

[97] Matthias Neubauer and Peter Thiemann. An implementation of session types. In *PADL '04*, volume 3057 of *Springer LNCS*, pages 56–70, 2004.

[98] Rumyana Neykova, Raymond Hu, Nobuko Yoshida, and Fahd Abdeljallal. A session type provider: compile-time API generation of distributed protocols with refinements in f#. In *CC*, pages 128–138. ACM, 2018.

[99] Rumyana Neykova and Nobuko Yoshida. Multiparty session actors. In *COORDINATION '14*, pages 131–146, 2014.

[100] Rumyana Neykova and Nobuko Yoshida. Let it recover: multiparty protocol-induced recovery. In *CC*, pages 98–108. ACM, 2017.

[101] Rumyana Neykova, Nobuko Yoshida, and Raymond Hu. SPY: local verification of global protocols. In *RV '13*, volume 8174 of *Springer LNCS*, pages 358–363, 2013.

[102] Nicholas Ng, José Gabriel de Figueiredo Coutinho, and Nobuko Yoshida. Protocols by default - safe MPI code generation based on session types. In *CC '15*, pages 212–232, 2015.

[103] Nicholas Ng, Nobuko Yoshida, and Kohei Honda. Multiparty session C: safe parallel programming with message optimisation. In *TOOLS '12*, pages 202–218, 2012.

[104] Nicholas Ng, Nobuko Yoshida, Olivier Pernet, Raymond Hu, and Yiannos Kryftis. Safe parallel programming with Session Java. In *COORDINATION '11*, volume 6721 of *Springer LNCS*, pages 110–126, 2011.

[105] Jesper Öqvist. Extendj: extensible java compiler. In *Programming*, pages 234–235. ACM, 2018.

[106] Dominic A. Orchard and Tomas Petricek. Embedding effect systems in haskell. In *Haskell*, pages 13–24. ACM, 2014.

[107] Dominic A. Orchard and Nobuko Yoshida. Effects as sessions, sessions as effects. In *POPL*, pages 568–581. ACM, 2016.

[108] Luca Padovani. Fair subtyping for multi-party session types. In *COORDINATION*, volume 6721 of *Lecture Notes in Computer Science*, pages 127–141. Springer, 2011.

[109] Luca Padovani. Fair subtyping for open session types. In *ICALP (2)*, volume 7966 of *Lecture Notes in Computer Science*, pages 373–384. Springer, 2013.

[110] Luca Padovani. Type reconstruction for the linear π-calculus with composite regular types. *Logical Methods in Computer Science*, 11(4), 2015.

[111] Luca Padovani. Fair subtyping for multi-party session types. *Mathematical Structures in Computer Science*, 26(3):424–464, 2016.

[112] Luca Padovani. Context-free session type inference. In *ESOP*, volume 10201 of *Lecture Notes in Computer Science*, pages 804–830. Springer, 2017.

[113] Luca Padovani. A simple library implementation of binary sessions. *J. Funct. Program.*, 27:e4, 2017.

[114] Tomas Petricek, Gustavo Guerra, and Don Syme. Types from data: making structured data first-class citizens in f#. In *PLDI*, pages 477–490. ACM, 2016.

[115] Frank Pfenning and Dennis Griffith. Polarized substructural session types. In *FoSSaCS*, volume 9034 of *Lecture Notes in Computer Science*, pages 3–22. Springer, 2015.

[116] Benjamin C. Pierce. *Types and programming languages*. MIT Press, 2002.

[117] Gordon D. Plotkin. LCF considered as a programming language. *Theor. Comput. Sci.*, 5(3):223–255, 1977.

[118] Jeff Polakow. Embedding a full linear lambda calculus in haskell. In *Haskell*, pages 177–188. ACM, 2015.

[119] Jon Postel and Joyce K. Reynolds. File transfer protocol. *RFC*, 959:1–69, 1985.

[120] Klaas Pruiksma and Frank Pfenning. A message-passing interpretation of adjoint logic. In *PLACES*, volume 291 of *Electronic Proceedings in Theoretical Computer Science*, pages 60–79. Open Publishing Association, 2019.

[121] Riccardo Pucella and Jesse A. Tov. Haskell session types with (almost) no class. In *Proceedings of the 1st ACM SIGPLAN Symposium on Haskell*, pages 25–36. ACM Press, 2008.

[122] Alceste Scalas, Ornela Dardha, Raymond Hu, and Nobuko Yoshida. A linear decomposition of multiparty sessions for safe distributed programming. In *ECOOP*, volume 74 of *LIPIcs*, pages 24:1–24:31. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2017.

[123] Alceste Scalas, Ornela Dardha, Raymond Hu, and Nobuko Yoshida. A linear decomposition of multiparty sessions for safe distributed programming. Technical Report 2, Imperial College London, 2017.

[124] Alceste Scalas and Nobuko Yoshida. Lightweight session programming in scala. In *ECOOP*, volume 56 of *LIPIcs*, pages 21:1–21:28. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

[125] Alceste Scalas and Nobuko Yoshida. Less is more: multiparty session types revisited. *Proceedings of the ACM on Programming Languages*, 3(POPL):30, 2019.

[126] Scribble project homepage. www.scribble.org.

[127] Robert E. Strom and Shaula Yemini. Typestate: A programming language concept for enhancing software reliability. *IEEE Trans. Softw. Eng.*, 12(1):157–171, 1986.

[128] Joshua Sunshine, Karl Naden, Sven Stork, Jonathan Aldrich, and Éric Tanter. First-class state change in plaid. In *OOPSLA*, pages 713–732. ACM, 2011.

[129] Asumu Takikawa, Daniel Feltey, Ben Greenman, Max S. New, Jan Vitek, and Matthias Felleisen. Is sound gradual typing dead? In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '16, pages 456–468. ACM, 2016.

[130] Peter Thiemann and Vasco T. Vasconcelos. Context-free session types. In *ICFP*, pages 462–475. ACM, 2016.

[131] Bernardo Toninho, Luís Caires, and Frank Pfenning. Higher-order processes, functions, and sessions: A monadic integration. In *ESOP*, volume 7792 of *Lecture Notes in Computer Science*, pages 350–369. Springer, 2013.

[132] Antonio Vallecillo, Vasco T. Vasconcelos, and António Ravara. Typing the behavior of software components using session types. *Fundamenta Informaticæ*, 73(4):583–598, 2006.

[133] Vasco T. Vasconcelos. Fundamentals of session types. *Inf. Comput.*, 217:52–70, 2012.

[134] Vasco T. Vasconcelos, Simon J. Gay, and António Ravara. Typechecking a multithreaded functional language with session types. *Theoret. Comp. Sci.*, 368(1–2):64–87, 2006.

[135] Vasco Thudichum Vasconcelos. Fundamentals of session types. In *SFM*, volume 5569 of *Lecture Notes in Computer Science*, pages 158–186. Springer, 2009.

[136] A. Laura Voinea, Ornela Dardha, and Simon J. Gay. Resource Sharing via Capability-Based Multiparty Session Types. In *IFM*, volume 11918 of *Lecture Notes in Computer Science*, pages 437–455. Springer, 2019.

[137] A. Laura Voinea, Ornela Dardha, and Simon J. Gay. Typechecking Java Protocols with [St]Mungo. In *FORTE*, volume 12136 of *Lecture Notes in Computer Science*, pages 208–224. Springer, 2020.

[138] A Laura Voinea and Simon J Gay. Benefits of session types for software development. In *Proceedings of the 7th International Workshop on Evaluation and Usability of Programming Languages and Tools*, pages 26–29. ACM, 2016.

[139] Philip Wadler. Propositions as sessions. In *ICFP*, pages 273–286. ACM, 2012.

[140] David Walker and J. Gregory Morrisett. Alias types for recursive data structures. In *TIC*, LNCS, pages 177–206. Springer, 2000.

[141] Robert NM Watson, Simon W Moore, Peter Sewell, and Peter G Neumann. An introduction to cheri. Technical report, University of Cambridge, Computer Laboratory, 2019.

[142] Max Willsey, Rokhini Prabhu, and Frank Pfenning. Design and implementation of concurrent C0. In *LINEARITY*, volume 238 of *EPTCS*, pages 73–82, 2016.

[143] Roger Wolff, Ronald Garcia, Eric Tanter, and Jonathan Aldrich. Gradual typestate. In *ECOOP '11*, volume 6813 of *Springer LNCS*, pages 459–483, 2011.

[144] Nobuko Yoshida, Pierre-Malo Deniélou, Andi Bejleri, and Raymond Hu. Parameterised multiparty session types. In *FOSSACS*, 2010.

[145] Nobuko Yoshida, Pierre-Malo Deniélou, Andi Bejleri, and Raymond Hu. Parameterised multiparty session types. In *Proceedings of the 13th International Conference on Foundations of Software Science and Computational Structures*, FOSSACS'10, page 128–145, Berlin, Heidelberg, 2010. Springer-Verlag.

[146] Nobuko Yoshida, Raymond Hu, Rumyana Neykova, and Nicholas Ng. The scribble protocol language. In *International Symposium on Trustworthy Global Computing*, pages 22–41. Springer, 2013.