UNIVERSITÀ DEGLI STUDI DI TORINO

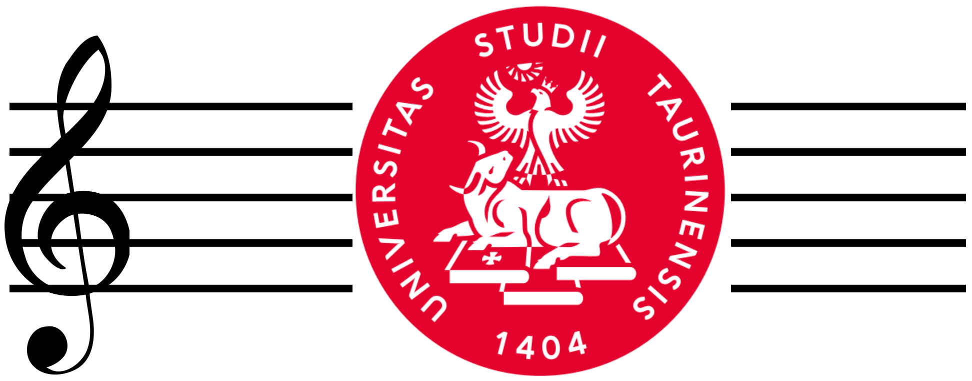
DIPARTIMENTO DI INFORMATICA

SCUOLA DI SCIENZE DELLA NATURA

Doctoral School in Computer Science - Cycle XXXV

Concerto Grosso *for* *Sessions*

Fair Termination of Sessions



Candidate
Luca Ciccone

Supervisors
Prof. Luca Padovani
Prof. Ugo de'Liguoro

Reviewers
Prof. Eugenio Moggi
Prof. Ornela Dardha

PhD Coordinator: Prof. Viviana Patti
Submitted on: December 2022

Gratias Vobis Ago

Institutional

To University of Torino and the entire PhD board 2020 and 2021 have been hard years for everyone. However, UniTo successfully managed to provide courses and side activities in the best way possible. At the beginning of 2020 TYPES conference should have been held in Torino but COVID was spreading in Italy. The conference moved to a virtual space. No one knew it was the first of a long sequence of remote events. PhD related events have been obviously affected as well. Nonetheless, courses have been well organized fast in a remote fashion.

To my supervisor Luca Padovani I skip the COVID part as it would be obvious and I move to mere professional considerations. As soon as my PhD started Luca and I discussed about the ways in which we could have taken advantage of the work I did during the master thesis. This way we immediately found the right intersection between by knowledge and my PhD project and we started working together. I never worked *for* but *with* him. Such an approach led to fast and satisfactory research outcomes. Moreover Luca always shared his connections with other research groups, *e.g.* at University of Glasgow. I have been his first PhD student so I hope that these lines will give him a feedback about his supervision.

To University of Genova and Elena Zucca My master thesis supervisor Elena Zucca agreed with Luca for giving me an office at DIBRIS, UniGe. This way we could start a fruitful research collaboration. During last year we managed to publish together a paper at ECOOP that earned the *distinguished paper* award. Hence, I'm grateful to both the University of Genova and Elena for allowing me to use such office.

To EuroProofNet, University of Glasgow and Ornela Dardha After two years of remote human interactions, in-person events started again. During the pandemic I have been in touch with Ornela Dardha whose research interests were very close to mine. Unfortunately I could not have met her research group. In the meantime EuroProofNet, a european project aimed at connecting researchers on formal proofs, was born. Such project gave me the funds for making two visits to Ornela Dardha at University of Glasgow. I have been welcomed in the best way.

To the reviewers I'm grateful to Eugenio Moggi and Ornela Dardha for having spent a lot of time reading this thesis. They provided useful comments and important references. I hope this work will be useful for their research groups.

...a little flattery

To my super-visor Luca Padovani Coming to personal considerations, I am grateful to Luca for many reasons. In general, as an empathetic person I found his approach of sharing not only his ideas but also his feelings very inspiring. I learned how to face a variety of situations thanks to him, from the anxiety in meeting deadlines to the tricks in providing successful presentations. These skills will be helpful for my career. Last but not least, I'd like to thank Luca for his support during the pandemic when I had serious COVID problems in my family; we were writing the ICALP paper and the result has been successful.

To my unofficial supervisor Elena Zucca I'd like to express my gratitude to Elena for her precious suggestions during the PhD and before starting it. She put me in touch with Luca Padovani at the end of my master thesis. Although she was not officially reported as my co-supervisor, she demonstrated a high involvement in working together as well as with my colleague and friend Francesco Dagnino (UniGe). As for Luca, I'd like to thank her for her support when I had COVID problems in my family. When I was working on the ICALP paper with Luca, in the meantime I was writing the ITP one with Elena and Francesco. Such paper has been accepted and together with the PPDP one it introduced me to the formal proofs community.

To Ornela Dardha I had the opportunity of meeting Ornela only last year. She welcomed me in Glasgow as a friend more than as a professor. I immediately discovered that it was not just a special treatment for visitors as she has a very special way of dealing with her research group. Although I'm usually not so keen on those situations in which *colleagues* and *friends* automatically overlap, I'm sure that Ornela's approach is fantastic both for her students and visitors. The reason behind my opinion is that Ornela is a very empathetic person and she cares about the mood of others a lot. For this reason I decided to visit her two more times the same year and I really hope to meet her again even if I will not be involved in the research world anymore. *That's okay for today. Now you should go to the pub! (cit.)*

SPLS, 26th October 2022.

Personal

To my family This is a special paragraph. I'm usually in trouble when I have to write something to express my gratitude to my family as putting on paper deep feelings is a very hard task (and I also have to deal with my eyes filled with tears). What lies behind the usual sentence *for their support and blah blah*? I would like to completely change the point of view and try something new. Obviously, the focus is on my parents Renato Ciccone and Marirosa Gaggero. Let me start with a consideration: I trust stereotypes and in Genova we are famous as people that do not want to spend money. My studies were not free of course, so I'm very grateful to my parents for the sacrifices they made for paying a lot for so many years. Although I managed to win scolarships different times, I always used such money for whatever but paying university subscriptions; I was younger and naive. Furthermore, when I decided to start a PhD, I always asked myself whether such a title would have been useful for my career outside the academy. I honestly shared my doubts with my parents. In the end I found a satisfactory number of job opportunities positively influenced by these three years of research. At last, the years of PhD, and in particular the last one, made me grow up a lot as a person. Hence, I hope to have made my parents proud of the results.

To my grandmother Anna (Ciccone) Peruzzi Things become harder. She is a dresskmaker for children, she loves classical music, she attends theatres and music associations...more in general, she loves beauty in its essence. *What is a person without any cultural interest?*, she always asked. From her I learned the things to appreciate and study in my free time and that I deepened every day. I'm grateful to her for her ambitions towards me that I hope to have satisfied.

To my ggf Ruggero Peruzzi He was born on 19 Novemeber, 1891. He was an antiquarian in Genova. My grandmother learned from him the love of beauty. During a journey he met the painter C. Tafuri and he paid for him an office in Palazzo della Borsa in Genova. During WW2 he has been captured by fascists as they wanted the names of some partisans. He never told such information and fascists started torturing him. He had only one opportunity to say goodbye to her daughter. He has been killed by fascists and his body has been found on 18 February 1945 in front of Brignole train station in Genova where you can find his tombstone.

Ruggero Peruzzi after Edmé Bouchardon.

Concerto Grosso? This last acknowledgment goes to A. Schnittke, A. Schönberg, A. Berg, J. S. Bach, B. Galuppi, A. Pärt as well as to many other great composers. In particular, Schnittke's music has been the soundtrack of the last three years. His *Concerti Grossi* obviously recall baroque music but at the same time they are progressively cracked and filled with XXth century influences.

Overture

Sessions are a fundamental notion in message-passing systems. A session is an abstract notion of communication between parties where each one owns an endpoint. *Session types* are types that are assigned to the endpoints and that are used to statically and dynamically enforce some desired properties of the communications, such as the absence of deadlocks.

Properties of concurrent systems are usually divided in *safety* and *liveness* ones and depending on the class it belongs to, a property is defined using different (dual) techniques. However, there exist some properties that require to mix both techniques and the challenges in defining them are exacerbated in *proof assistants* (*e.g.* Agda, Coq, . . .), that is, tools that allow users to formally characterize and prove theorems. In particular, we mechanize the meta-theory of *inference systems* in Agda.

Among the interesting properties that can be studied in the session-based context, we study *fair termination* which is the property of those sessions that can always eventually reach *successful* termination under a *fairness* assumption. Fair termination implies many desirable and well known properties, such as lock freedom. Moreover, a lock free session does not imply that other sessions are lock free as well. On the other hand, if we consider a session and we assume that all the other ones are fairly terminating, we can conclude that the one under analysis is fairly terminating as well.

Contents

Praeludium

Nowadays concurrent and distributed systems are ubiquitous in the computer science world. It is not surprising that there exists a wide research about how to enforce some desired properties of communications in such systems.

A Gentle Introduction

Sessions The π-calculus is a process calculus used for modeling concurrent systems and it is based on *channels* that are owned by the entities involved in the communication and that are used to exchange messages. Session types [Honda, 1993] are a formalism to describe communication protocols. A session is represented by a channel that connects processes and session types are assigned to them. They describe how the channels will be used by the processes that own them. Session types have been originally introduced to model the interaction between *two* entities and they have been later generalized to multiparty [Honda et al., 2008]. There exists a broad family of desired properties that one would like to enforce in a communication. Among them, the most common ones are that messages are exchanged in the right order (*protocol fidelity*) or that there are no communication errors (*communication safety*).

Safety vs Liveness - Fair Termination In general, properties can be divided in *safety* (nothing bad ever happens) and *liveness* (something good will eventually happen) ones [Owicki and Lamport, 1982]. According to the class to which it belongs, a property is defined using different and dual techniques: *induction* for safety and *coinduction* for liveness. Actually, there exist some properties that fall in between as they combine safety and liveness aspects.

We can informally describe them using the motto "something good always eventually happens" and their dual. Hence, such properties are hard to deal with as they mix induction and coinduction. In a communication-based scenario, protocol fidelity is a safety property and the eventual termination of a session is a liveness one. In this thesis we study *fair termination* which is the property of those sessions that can always eventually (successfully) terminate under a fairness assumption. As the reader might guess, fair termination joins a safety and a liveness part. We investigate such property as it entails many well knows properties that are studied in the literature.

Fair Subtyping Subtyping between session types is a fundamental relation in this thesis. It induces a substitution principle [Liskov and Wing, 1994] which states that a process that uses an endpoint according to some type can be substituted with another process that uses the endpoint according to a supertype. Although the principle seems in the wrong direction it is proved to be equivalent to the right-to-left variant by taking into account endpoints instead of processes [Gay, 2016]. The original subtyping relation [Gay and Hole, 2005] is studied to preserve the safety properties of a session but not the livess ones. Indeed an application of the original subtyping can break the termination property of the involved session. In this thesis we rely on *fair subtyping* [Padovani, 2014, 2016] which is a liveness-preserving variant of the subtyping in Gay and Hole [2005].

(Generalized) Inference Systems Inference systems are a widely used framework in computer science for defining predicates by means of (meta) rules. They support both inductive and coinductive definitions and they have been recently generalized to deal with those predcates that mix the two approaches [Ancona et al., 2017b]. Roughly, an inference system, that is, a set of (meta)rules, is interpreted either inductively or coinductively. In a generalized inference system the key ingredient is the addition of (meta)corules and the interpretation is obtained by properly mixing the inductive and the coinductive one. Then, inference systems also provide *proof principles* for proving the correctness of a definition with respect to a semantic one called *specification* in a canonical way. Correctness is usually expressed in terms of *soundness* and *completeness*. The reference principles are *induction*, *coinduction* which involve inductive and coinductive predicates, respectively. *Bounded coinduction* is a generalization of coinduction and is used when dealing with definitions through generalized inference systems. We will rely on (generalized) inference systems for defining most of the notions we will introduce.

Proof Assistants These days there is an increasing interest in *theorem provers*, that is, tools that allow users to write proofs by relying on a language that resembles a programming language. The advantage of using proof assistants is that the proofs are certified (provided that the tool is itself consistent). Due to the novelty of the research field, there is a focus in the literature in providing big libraries with the aim of making the community agree on some design techniques. For example, when formalizing calculi, De Bruijn indices are usually preferred for dealing with variables. Many research communities are starting to rely on theorem provers to mechanize existing theories. The community working on behavioral types is affected by this phenomenon as well. Moreover, apart from traditional conferences on formal proofs (*e.g.* CPP, ITP), a very first workshop on *Verification of Session Types* has been organized in 2020 and 2021. The main output of such event was the lack of a reference methodology for mechanizing session-based theories. In this thesis we will refer to the Agda proof assistant [The Agda Team, 2022]. Notably, we refer to theorem provers and proof assistant interchangeably.

Agda Agda [The Agda Team, 2022] is an interactive proof assistant based on intuitionistic logic. The interactivity aspect is fundamental as the user is guided step-by-step while mechanizing a proof. The tool can be download for different platforms and comes with only the very basic definitions, *e.g.* natural numbers. The *standard library* can be added separately and it is constantly updated in order to keep the modules consistent with the latest version of Agda. Moreover, Agda has built-in support for both inductive and coinductive predicates. In Ciccone [2020] we made an in-depth analysis of all the techniques that users can follow when mechanizing predicates and we found out that pure (co)inductive inference system have a natural and intuitive encoding in Agda. However, for what concerns those predicates defined by generalized inference systems, there is no support. Indeed, although a generalized inference system can still be defined, the resulting code becomes hard to understand and it contains many duplicated notions.

Contributions

Can well-typed programs go well?

Developing type systems for enforcing *liveness* properties of processes is not an easy task. For example, in the context of this thesis the type systems of Kobayashi [2002a], Padovani [2014] adopt a sophisticated technique for guaranteeing that all pending actions will be eventually executed. The question we want to answer is in sharp contrast with the famous Robin Milner's sentence "Well typed programs cannot go wrong" [Milner, 1978] which points out the fact that type systems are usually made for enforcing *safety* aspects.

Fair Termination We present type systems for **enforcing fair termination** in three different scenarios. First, we deal with *binary* sessions, that is, sessions involving only two participants. This base case is fundamental to highlight all the technical aspects and additional properties that the type system must ensure in order to guarantee fair termination. Then, we generalize such type system to the *multiparty* case. Although many notions can be easily adapted from the binary case, there are some technical details that are hard to deal with. For what concerns the additional properties that the type system has to enforce, we can still rely on the binary type system since binary sessions are the simplest multiparty ones. At last, we show a type for enforcing fair termination in the linear π-calculus. This last part falls in between the previous type systems and those type systems interpreting session-based calculi in the context of linear logic. The interesting difference between the type systems for sessions and the one for the π-calculus is that in the second there is no notion of fair subtyping.

Can Agda support definitions mixing induction and coinduction?
Can Agda proofs match the pen-and-paper ones?

As we mentioned before, Agda supports purely (co)inductive definitions. Moreover, a predicate defined using an inference system can be naturally encoded. On the other hand, when we deal with a predicate that involves both induction and coinduction, there is no built-in support [Ciccone, 2020]. As a result, we can still mechanize such predicates but the resulting code contains many duplicated notions and it differs from the definition given through a generalized inference system. Another drawback of mechanizing notions in Agda is that the proofs become very hard to be understood as they differ from the hand written ones since they require many additional lemmas related to the technicalities of the proof assistant, *e.g.* equalities.

Hence, we first ask whether it is possible to **encode generalized inference systems in Agda in a natural way**. Then we ask if we can “hide” all the Agda related technicalities so that the **proofs are shown to the user in a more friendly flavor**.

An Agda Library for Inference Systems (Generalized) inference systems are used throughout the entire thesis. Since we aimed at mechanizing some results in Agda, we thought about a way for supporting the framework in the proof assistant. For this sake we present a **library** that allows users to formalize inference systems using a syntax that resembles the one used on paper. Moreover, the library supports the usage of corules that are not built-in supported (see Ciccone [2020] for more details). Clearly, all the proof principles are encoded as well. As an important and useful side effect, the codes of the proofs based on the library resemble the proofs written on paper. We think that this contribution can be very helpful for the community since closed proofs are usually very hard to read and understand.

Mechanizing Properties of Session Types We take advantage of the library we developed to **mechanize some notions in the session types context**. We first mechanize an alternative definition of session types that aims at simplifying the codes as much as possible. Such characterization has also the advantage of supporting *dependency* with respect to previously exchanged messages. Then we focus on three interesting properties of session types and we characterize them using generalized inference systems: fair termination of a session type, fair compliance (successful fair termination) of a binary session and fair subtyping. We mechanize the three inference system as well as their correctness proofs with respect to the semantic definitions. As mentioned before, the community working on session types is increasingly interested in proof assistants but so far there is a lack of a reference methodology. We hope that our results and our experience will be used for finding a guideline on which all the community agrees.

Structure of the Thesis

Concerto Grosso As the title might suggest, the thesis is structured in three parts as the original italian *Concerto Grosso* form of baroque music. Moreover, since binary sessions are the simplest multiparty ones, we will jump between the two scenarios according to our needs. In general, we will show paradigmatic examples in the binary setting and we will prove the main notions in the more general multiparty one. The continuous alternation of communications between two and more participants resembles again the *Concerto Grosso* which is based on a dialogue between a small ensemble and the full orchestra. The tempo indications at the beginning of each part are consistent with the fast-slow-fast schema of the concerto grosso and we use them to jokingly indicate that the part that should be read more carefully is the central one. At last, in order to help the reader we use treble, soprano and bass clefs to denote texts inside chapters, sections and subsections, respectively.

Part I: Key Notions There are different notions that we will use throughout Parts II and III. Thus, we decided to dedicate the first part to introduce all the main ingredients of the thesis. At the beginning we discuss the usual classification of properties in safety and liveness ones in Section 1.1. Then, in Section 1.2 we introduce (Generalized) inference systems as a reference framework and in Section 1.3 we introduce syntax and semantics of binary and multiparty session types. We dedicate Chapter 2 to explain what is *fair termination* and to express it in a general setting, that is, considering an arbitrary transition system. Then, we instantiate the property in the session types context. At last, in Chapter 3 we focus on the subtyping relation between session types. We first introduce the original subtyping relation [Gay and Hole, 2005] and we show why it can break the liveness properties on the session on which it is applied. We then introduce *fair subtyping* as a liveness-preserving refinement of the original relation. We present fair subtyping in two different flavours; first using a purely coinductive inference system and then using a generalized inference system.

Part II: Type Systems This part is dedicated to the study of the three type systems for fair termination and it is split according to the calculus under analysis. In Chapter 4 we present a type system for a calculus with binary sessions. Chapter 5 generalizes such type system to the multiparty case. At last, in Chapter 6 we show a type system for a π-calculus that is based on linear logic. The three chapters are stuctured similarly. We first introduce the calculus (and the types) and then we show the typing rules. We dedicate the rest of each chapter to detail the soundness proofs. As we mentioned before, in Chapter 4 we focus on those paradigmatic examples that point out the additional properties that the type system must enforce. Since such examples hold in the multiparty setting as well, in Chapter 5 we present some more involved processes.

Part III: Agda Mechanizations This third and last part focuses on the mechanization aspects. In Chapter 7 we present the library for supporting in Agda generalized inference systems. We first mechanize the meta theory of inference systems as well as the proof principles and then we show some basic examples of usage. Then in Chapter 8 we present the definitions through generalized inference systems of three properties of session types: fair termination of a session type, fair compliance of a binary session and fair subtyping. Notably, we mechanize the general notion of *fair termination* and we relate it to the semantic definitions of the first two properties. For each property we show the mechanization of the inference system and we show how proof principles are used in the correctness proofs. At last, we detail the proofs of fair compliance.

Published Works

We conclude the introduction by relating the content of the thesis with the published works of the author.

Fair Termination Chapter 4 is a refined version of Ciccone and Padovani [2022c]. Differently from Ciccone and Padovani [2022c], here we adopt the new characterization of fair subtyping that we introduced later in Ciccone et al. [2022]. Chapter 5, Chapter 6 show the type systems that have been presented in Ciccone et al. [2022] and in Ciccone and Padovani [2022b], respectively.

Agda The Agda library for (Generalized) Inference Systems in Chapter 7 has been presented in Ciccone et al. [2021a] which is based on Ciccone [2020]. At last, the Agda formalizations of the properties of session types in Chapter 8 have been presented in Ciccone and Padovani [2022a, 2021a].

Not in this thesis In Ciccone and Padovani [2020] we provide an Agda mechanization of a linear π-calculus with dependent types. One of the interesting features of such formalization is that we can inject Agda terms in the calculus by using a proper constructor for the terms of the calculus. Although that work covers some topics that are related to the present thesis (*e.g.* π-calculus, Agda mechanization, induction/coinduction), we do not give details about it as it falls outside the core topic the thesis. Moreover, we are working on an Agda unification library and on co-contextual typing inference algorithms for the linear and shared *pi*-calculus proved to be correct in the proof assistant. Such work is based on Zalakain and Dardha [2021].

Part I

Key Notions
(Vivace)

Chapter 1

Introductory Notions

This chapter is dedicated to the analysis of the main notions that are treated throughout the thesis. First, we recall the usual distinction of properties of concurrent systems in *safety* and *liveness* ones [Owicki and Lamport, 1982]. According to the class to which it belongs, a property must be defined using a different (dual) technique. The same happens when we want to prove the correctness of such definitions. Then, we introduce *Generalized Inference Systems* [Ancona et al., 2017b]. Inference systems are a well established framework in the literature to define purely (co)inductive predicates as they provide canonical techniques for proving the correctness of a definition. In this thesis we mainly focus on their generalization with *corules* in order to treat those predicates that require a mix of induction and coinduction. After that we move to the main topics of the thesis. We give an overview of *session types* by describing what such types are and what they are used for. In a few words, they are used to model the communication in message passing systems and to statically enforce some desired properties. They have been originally introduced by Honda [1993] in a binary context and later generalized by Honda et al. [2008] to *multiparty* to deal with those scenarios in which more entities participate to the communication. We then review some interesting desirable properties that fall in the intersection between safety and liveness ones.

The chapter is organized as follows. In Section 1.1 we review the usual classification of properties into safety and liveness ones and we show that some fall in the intersection of the two classes. In Section 1.2 we introduce generalized inference systems as a reference framework that can be used to deal with such kind of properties. At last, in Section 1.3 we introduce session types in both their binary and multiparty variants.

1.1 Safety & Liveness

It is well known that properties can be distinguished between *safety* and *liveness* one. Informally, the two classes are identified by the mottos “nothing bad ever happens” and “something good eventually happens” [Owicki and Lamport, 1982]. For example, in a network of communicating processes, the absence of communication errors and of deadlocks are safety properties, whereas the fact that a protocol or a process can always successfully terminate is a liveness property. Because of their different nature, characterizations and proofs of safety and liveness properties rely on fundamentally different (dual) techniques: safety properties are usually based on invariance (coinductive) arguments, whereas liveness properties are usually based on well foundedness (inductive) arguments [Alpern and Schneider, 1985, 1987b].

The correspondence and duality between safety/coinduction and liveness/induction is particularly apparent when properties are specified as formulas in the modal *μ-calculus* [Kozen, 1983, Stirling, 2001, Bradfield and Stirling, 2007], a modal logic equipped with least and greatest fixed points: safety properties are expressed in terms of greatest fixed points, so that the “bad things” are ruled out along all (possibly infinite) program executions; liveness properties are expressed in terms of least fixed points, so that the “good things” are always within reach along all program executions. Since the μ-calculus allows least and greatest fixed points to be interleaved arbitrarily, it makes it possible to express properties that combine safety and liveness aspects, although the resulting formulas are sometimes difficult to understand.

In the next examples we informally state some (co)inductive properties that we will characterize in details in Section 1.2 and that we will characterize in Agda in Section 7.1.

Example 1.1.1. “x belongs to the possibly infinite list l”

This is an example of a liveness property. Indeed, even if the list l is infinite, if x belongs to l, then we need to inspect only a finite portion of the list. Hence, induction is required. ⌟

Example 1.1.2. “All the numbers in the possibly infinite list l are positive”

This is an example of a safety property. Indeed, in order to actually state that a list is made of only positive numbers, we have to inspect it entirely. It is clear that coinduction is required since the list can be infinite. ⌟

Example 1.1.3. " x is the maximum element of the possibly infinite list l"
The predicate is composed by a safety and a liveness part.

- Safety*: x is greater that all the elements in l*
- Liveness*: x belongs to l*

Hence, this property requires a mix of induction and coinduction. ⌟

1.2 (Generalized) Inference Systems

A different way of specifying (and enforcing) properties is by means of *inference systems* [Aczel, 1977]. Inference systems admit two natural interpretations, an inductive and a coinductive one respectively corresponding to the least and the greatest fixed points of their associated inference operator. Unlike the μ-calculus, however, they lack the flexibility of mixing their different interpretations since the inference rules are interpreted either all inductively or all coinductively. For this reason, it is generally difficult to specify properties that combine safety and liveness aspects by means of a single inference system. *Generalized Inference Systems* (GISs) [Ancona et al., 2017b, Dagnino, 2019] admit a wider range of interpretations, including intermediate fixed points of the inference operator associated with the inference system different from the least or the greatest one. This is made possible by the presence of *corules*, whose purpose is to provide an inductive definition of a space within which a coinductive definition is used. Although GISs do not achieve the same flexibility of the modal μ-calculus in combining different fixed points, they allow for the specification of properties that can be expressed as the intersection of a least and a greatest fixed point. This feature of GISs resonates well with one of the fundamental results in model checking stating that every property can be decomposed into a conjunction of a safety property and a liveness one [Alpern and Schneider, 1985, 1987b, Baier and Katoen, 2008].

1.2.1 Definitions and interpretations

We first summarize the main notions and definitions of inference systems. Then we generalize them by introducing (meta)corules.

Given an inference system, one has to take an *interpretation* that identifies the set of defined judgments. When dealing with a generalized inference system, such interpretation is obtained by combining both the inductive and the coinductive ones and by taking into account corules in the proper way.

Definition 1.2.1 (Inference Systems & Rules)**.** An *inference system* [Aczel, 1977] $\mathcal{I}$ over a *universe* $\mathcal{U}$ of *judgments* is a set of *rules*, which are pairs $\langle pr, j\rangle$ where $pr \subseteq \mathcal{U}$ is the set of *premises* of the rule and $j \in \mathcal{U}$ is the *conclusion* of the rule. A rule without premises is called *axiom*. Rules are typically presented using the syntax

$$\frac{pr}{j}$$

where the line separates the premises (above the line) from the conclusion (below the line).

Since rules may be infinite, inference systems are usually described by using *meta-rules* written in a meta-language. Informally, meta-rules group rules according to their shape.

Definition 1.2.2 (Metarule)**.** A *meta-rule* ρ is a tuple $\langle \mathcal{C}, Pr, J, \Sigma\rangle$ where

- $\mathcal{C}$ is a set called *context*
- $Pr = \langle J_1, \ldots, J_n\rangle$ is a finite sequence of functions of type $\mathcal{C} \to \mathcal{U}$ called *(meta-)premises*
- J is a function of type $\mathcal{C} \to \mathcal{U}$ called *(meta-)conclusion*
- Σ is a subset of $\mathcal{C}$ called *side condition*

Given $\rho = \langle \mathcal{C}, \langle J_1, \ldots, J_n\rangle, J, \Sigma\rangle$, the inference system $\|\rho\|$ *denoted by* ρ is defined by:

$$\|\rho\| = \{\langle\{J_1(c), \ldots, J_n(c)\}, J(c)\rangle \mid c \in \Sigma\}$$

Definition 1.2.3 (Meta System)**.** An *inference meta-system* $\mathcal{I}$ is a set of meta-rules. The inference system $\|\mathcal{I}\|$ is the union of $\|\rho\|$ for all $\rho \in \mathcal{I}$.

A *predicate* on $\mathcal{U}$ is any subset of $\mathcal{U}$. An *interpretation* of an inference system $\mathcal{I}$ identifies a *predicate* on $\mathcal{U}$ whose elements are called *derivable judgments*.

To define the interpretation of an inference system $\mathcal{I}$, consider the *inference operator* associated with $\mathcal{I}$, which is the function $F_{\mathcal{I}} : \wp(\mathcal{U}) \to \wp(\mathcal{U})$ such that

$$F_{\mathcal{I}}(X) = \{j \in \mathcal{U} \mid \exists pr \subseteq X : \langle pr, j\rangle \in \mathcal{I}\}$$

for every $X \subseteq \mathcal{U}$. Intuitively, $F_{\mathcal{I}}(X)$ is the set of judgments that can be derived in one step from those in X by applying a rule of $\mathcal{I}$. Note that $F_{\mathcal{I}}$ is a monotone endofunction on the complete lattice $\wp(\mathcal{U})$, hence it has least and greatest fixed points. It can also be defined for a meta-rule ρ and an inference meta-system $\mathcal{I}$ as

$$F_\rho(X) = \{J(c) \mid c \in \Sigma, J_i(c) \in X \text{ for all } i \in 1..n\} \quad F_{\mathcal{I}}(X) = \textstyle\bigcup_{\rho\in\mathcal{I}} F_\rho(X)$$

Definition 1.2.4 ((Co)Inductive Interpretations)**.** The *inductive interpretation* $\mathsf{Ind}[\![\mathcal{I}]\!]$ of an inference system $\mathcal{I}$ is the least fixed point of $F_{\mathcal{I}}$ and the *coinductive interpretation* $\mathsf{CoInd}[\![\mathcal{I}]\!]$ is the greatest one.

From a proof theoretical point of view, $\mathsf{Ind}[\![\mathcal{I}]\!]$ and $\mathsf{CoInd}[\![\mathcal{I}]\!]$ are the sets of judgments derivable with well-founded and non-well-founded proof trees, respectively.

Generalized Inference Systems enable the definition of (some) predicates for which neither the inductive interpretation nor the coinductive one give the expected meaning.

Definition 1.2.5 (Generalized Inference System)**.** A *generalized inference system* is a pair $\langle\mathcal{I}, \mathcal{I}_{\mathsf{co}}\rangle$ where $\mathcal{I}$ and $\mathcal{I}_{\mathsf{co}}$ are inference systems (over the same $\mathcal{U}$) whose elements are called *rules* and *corules*, respectively. The interpretation of a generalized inference system $\langle\mathcal{I}, \mathcal{I}_{\mathsf{co}}\rangle$, denoted by $\mathsf{Gen}[\![\mathcal{I}, \mathcal{I}_{\mathsf{co}}]\!]$, is the greatest post-fixed point of $F_{\mathcal{I}}$ that is included in $\mathsf{Ind}[\![\mathcal{I} \cup \mathcal{I}_{\mathsf{co}}]\!]$.

Note that meta-corules are usually represented with a thicker line.

From a proof theoretical point of view, a GIS $\langle\mathcal{I}, \mathcal{I}_{\mathsf{co}}\rangle$ identifies those judgments derivable with an arbitrary (not necessarily well-founded) proof tree in $\mathcal{I}$ and whose nodes (the judgments occurring in the proof tree) are derivable with a well-founded proof tree in $\mathcal{I} \cup \mathcal{I}_{\mathsf{co}}$. For more details we refer to Dagnino [2019].

1.2.2 Proving correctness - Proof principles

𝄢 The advantage of using inference systems is that they provide canonical techniques for proving the correctness of a given definition. Consider now a *specification* $\mathcal{S} \subseteq \mathcal{U}$, that is an arbitrary subset of $\mathcal{U}$. We can relate $\mathcal{S}$ to the interpretation of a (generalized) inference system using one of the following proof principles. The *induction principle* [Sangiorgi, 2011, Corollary 2.4.3] allows us to prove the *soundness* of an inductively defined predicate by showing that $\mathcal{S}$ is *closed* with respect to $\mathcal{I}$. That is, whenever the premises of a rule of $\mathcal{I}$ are all in $\mathcal{S}$, then the conclusion of the rule is also in $\mathcal{S}$.

Proposition 1.2.1 (Induction). *$F_{\mathcal{I}}(\mathcal{S}) \subseteq \mathcal{S}$ implies $\mathsf{Ind}[\![\mathcal{I}]\!] \subseteq \mathcal{S}$.*

The *coinduction principle* [Sangiorgi, 2011, Corollary 2.4.3] allows us to prove the *completeness* of a coinductively defined predicate by showing that $\mathcal{S}$ is *consistent* with respect to $\mathcal{I}$. That is, every judgment of $\mathcal{S}$ is the conclusion of a rule whose premises are also in $\mathcal{S}$.

Proposition 1.2.2 (Coinduction). *$\mathcal{S} \subseteq F_{\mathcal{I}}(\mathcal{S})$ implies $\mathcal{S} \subseteq \mathsf{CoInd}[\![\mathcal{I}]\!]$.*

The *bounded coinduction principle* [Ancona et al., 2017b] allows us to prove the *completeness* of a predicate defined by a generalized inference system $\langle \mathcal{I}, \mathcal{I}_{\mathsf{co}} \rangle$. In this case, one needs to show not only that $\mathcal{S}$ is consistent with respect to $\mathcal{I}$, but also that $\mathcal{S}$ is *bounded* by the inductive interpretation of the inference system $\mathcal{I} \cup \mathcal{I}_{\mathsf{co}}$. Formally:

Proposition 1.2.3 (Bounded Coinduction). *$\mathcal{S} \subseteq \mathsf{Ind}[\![\mathcal{I} \cup \mathcal{I}_{\mathsf{co}}]\!]$ and $\mathcal{S} \subseteq F_{\mathcal{I}}(\mathcal{S})$ imply $\mathcal{S} \subseteq \mathsf{Gen}[\![\mathcal{I}, \mathcal{I}_{\mathsf{co}}]\!]$.*

Proving the boundedness of $\mathcal{S}$ amounts to proving the completeness of $\mathcal{I} \cup \mathcal{I}_{\mathsf{co}}$ (inductively interpreted) with respect to $\mathcal{S}$.

1.2.3 Examples

𝄢 We now characterize the examples from Section 1.1. For the sake of simplicity we only present the definitions and we informally state where the proof principles are used to prove the correctness (see Ciccone [2020] for the detailed proofs). In Section 7.1 we will show Agda mechanizations of all the definitions. In the following we write $xs[i]$ for the i-th element of list xs.

Example 1.2.1. *Recall the inductive property from Example 1.1.1. Consider the inference system $\mathcal{I}$*

$$\frac{}{member(x, x :: xs)} \qquad\qquad \frac{member(x, xs)}{member(x, y :: xs)}$$

where the axiom states that x is inside the list containing only x while the rule tells that, if x is inside xs, then it is in the same list with an additional element y. Such inference system $\mathcal{I}$ must be inductively interpreted*. Hence, we consider only those finite proof trees whose leaves are applications of the axiom.*

Let $\mathcal{S} = \{(x, xs) \mid \exists i \in \mathbb{N}.xs[i] = x\}$. We use the induction principle *to prove the* soundness *of the definition (*i.e. $\mathsf{Ind}[\![\mathcal{I}]\!] \subseteq \mathcal{S}$*).* ⌟

Example 1.2.2. *Recall the coinductive property from Example 1.1.2. Consider the inference system $\mathcal{I}$*

$$\frac{}{\mathit{allPos}\,[]} \qquad\qquad \frac{\mathit{allPos}\ xs}{\mathit{allPos}\ x :: xs}\ x > 0$$

where the axiom tells that the predicate trivially holds on the empty list and the rule states that, if a list xs is made of strictly positive numbers, then the predicate holds on such list with an additional element x provided that $x > 0$. Such inference system $\mathcal{I}$ must be coinductively interpreted*. Hence, we consider either those finite proof trees whose leaves are the axiom or the infinite ones that are obtained by applying infinitely many times the rule. Note that is not compulsory to interpret the rule coinductively. On the other hand, if we interpret it inductively we obtain the expected predicate if and only if the lists under analysis are finite.*

Let $\mathcal{S} = \{xs \mid \forall i \in \mathbb{N}.xs[i] > 0\}$. We use the coinduction principle *to prove the* completeness *of the definition (*i.e. $\mathcal{S} \subseteq \mathsf{CoInd}[\![\mathcal{I}]\!]$*).* ⌟

Example 1.2.3. *Recall the property from Example 1.1.3. We first consider the following inference system $\mathcal{I}$*

$$\frac{}{\mathit{maxElem}(x, x :: [])} \qquad\qquad \frac{\mathit{maxElem}(x, xs)}{\mathit{maxElem}(\mathit{max}(x, y), y :: xs)}$$

where the axiom states that x is the maximum of the list that contains only x, while the rule states that, if x is the maximum of xs, then, when we add a new number y to xs, the maximum becomes the greatest between x and y. Such inference system cannot be inductively interpreted since we have to entirely inspect a possibly infinite list. On the other hand, the coinductive interpretation does not give us the expected meaning. Indeed, for example we can derive $\mathit{maxElem}(2, xs)$ where $xs = 1 :: xs$ and 2 actually does not belong *to xs. The infinite proof tree is obtained by applying the rule infinitely many times.*

$$\cfrac{\cfrac{\vdots}{\mathit{maxElem}(2, \mathit{xs})}}{\mathit{maxElem}(2, 1 :: \mathit{xs})}$$

Hence, $\mathcal{I}$ characterize only the safety *component of maxElem which states that the maximum must be greater than all the elements in the list.*

Now we consider the following $\mathcal{I}_{\mathsf{co}}$ consisting of a single coaxiom

$$\frac{}{\mathit{maxElem}(x, x :: \mathit{xs})}$$

Such coaxiom forces the membership of the maximum into the list. The interpretation $\mathsf{Gen}[\![\mathcal{I}, \mathcal{I}_{\mathsf{co}}]\!]$ characterize the expected predicate. If we manage to find a finite proof tree in the inductive interpretation of the inference system with the additional coaxiom it means that the maximum belongs to the list.

Let $\mathcal{S} = \{(x, \mathit{xs}) \mid \forall i \in \mathbb{N}. x \geq \mathit{xs}[i] \text{ and } \mathit{member}(x, \mathit{xs})\}$. We use the bounded coinduction principle *to prove the* completeness *of the definition (*i.e. $\mathcal{S} \subseteq \mathsf{Gen}[\![\mathcal{I}, \mathcal{I}_{\mathsf{co}}]\!]$*).* ⌟

As noted at the beginning of the section, all the proofs of the examples are detailed in [Ciccone, 2020].

1.3 Session Types

Session types have been originally introduced by Honda [1993]. A session type describes the communication protocol that takes place over a channel, namely the allowed sequences of input/output actions performed by a process on that channel. This way it is possible to statically guarantee some desirable properties of the communication such as *protocol fidelity*, *communication safety* and *deadlock freedom.* Original session types concerned with the communication between two entities (*binary*) where each one owns an endpoint of the channel. They have been generalized to *multiparty* by Honda et al. [2008] to model the interaction between multiple participants.

1.3.1 Binary session types

We use *polarities* $\pi \in \{?, !\}$ to distinguish *input actions* (?) from *output actions* (!) and we write $\overline{\pi}$ for the *opposite* or *dual* polarity of π so that $\overline{?} = !$ and $\overline{!} = ?$. We use m to denote an element of a given set of *message tags* which may include values with a specific interpretation such as booleans, natural numbers, and so forth. The different terminology for labels is needed to avoid confusion with the labels used in the operational semantics.

Definition 1.3.1 (Binary Session Types)**.** Session types [Honda, 1993] are the possibly infinite, regular trees [Courcelle, 1983] coinductively generated by the grammar

$$S, T, U, V \ ::= \pi\mathsf{end} \mid \sum_{i \in I} \pi\mathsf{m}_i.S_i \mid \pi S.T$$

Session types of the form $\pi\mathsf{end}$ describe channels used for exchanging a session termination signal and on which no further communication takes place. Session types of the form $\pi U.S$ describe channels used for exchanging another channel of type U and then behaving according to S. Finally, session types of the form $\sum_{i \in I} \pi\mathsf{m}_i.S_i$ describe channels used for exchanging a tag m_k and then behaving according to S_k. Session types of the form $\sum_{i \in I} ?\mathsf{m}_i.S_i$ and $\sum_{i \in I} !\mathsf{m}_i.S_i$ are sometimes referred to as *external* and *internal* choices respectively, to emphasize that the label/tag being received or sent is always chosen by the sender process. In a session type $\sum_{i \in I} \pi\mathsf{m}_i.S_i$ we assume that I is not empty and that $i \neq j$ implies $\mathsf{m}_i \neq \mathsf{m}_j$ for every $i, j \in I$. Note that I is not necessarily finite, although regularity implies that there must be finitely many *distinct* S_i.

To improve readability we write $\pi\mathsf{m}.S$ when I is the singleton $\{\mathsf{m}\}$ (when the choice is trivial) and we define the partial $+$ such that

$$\textstyle\sum_{i \in I} \pi\mathsf{m}_i.S_i + \sum_{i \in J} \pi\mathsf{m}_i.S_i \quad = \quad \sum_{i \in I \cup J} \pi\mathsf{m}_i.S_i$$

when $I, J \neq \emptyset$ and $I \cap J = \emptyset$. Hereafter we specify possibly infinite session types by means of equations $S = \dots$ where the right hand side of the equation may contain guarded occurrences of the metavariable S. Guardedness guarantees that a session type S satisfying the equation exists and is unique [Courcelle, 1983].

We equip session types with a *labeled transition system* (LTS) that allows us to describe, at the type level, the sequences of actions performed by a process on a channel. We distinguish two kinds of transitions: *unobservable*

$$\textsc{lb-channel}\quad \frac{}{\pi U.S \xrightarrow{\pi U} S}$$

$$\textsc{lb-pick}\quad \frac{}{\sum_{i\in I} !\mathsf{m}_i.S_i \xrightarrow{\tau} !\mathsf{m}_k.S_k}\; k \in I \qquad \textsc{lb-tag}\quad \frac{}{\sum_{i\in I} \pi\mathsf{m}_i.S_i \xrightarrow{\pi\mathsf{m}_k} S_k}\; k \in I$$

$$\textsc{lb-tau-l}\quad \frac{S \xrightarrow{\tau} S'}{S \mid T \xrightarrow{\tau} S' \mid T} \qquad \textsc{lb-tau-r}\quad \frac{T \xrightarrow{\tau} T'}{S \mid T \xrightarrow{\tau} S \mid T'} \qquad \textsc{lb-sync}\quad \frac{S \xrightarrow{\overline{\alpha}} S' \qquad T \xrightarrow{\alpha} T'}{S \mid T \xrightarrow{\tau} S' \mid T'}$$

Figure 1.1: Labeled Transition System for Binary Session Types

transitions $S \xrightarrow{\tau} T$ are made autonomously by the process; *observable transitions* $S \xrightarrow{\alpha} T$ are made by the process in cooperation with the one it is interacting with through the channel. The label α describes the kind of interaction and has either the form πU (indicating the exchange of a channel of type U) or the form $\pi\mathsf{m}$ (indicating the exchange of tag m). The polarity π indicates whether the message is received (?) or sent (!). If we use a term of the form $S \mid T$ to describe the binary session as a whole, with the two interacting processes behaving as S and T, then we can formalize their interaction (at the type level) using the LTS for session types and the reduction rules in Figure 1.1.

We write $\overline{\alpha}$ for the dual of α, obtained by changing the polarity of α with the opposite one. A reduction occurs whenever one of the connected processes performs an unobservable transition ([lb-tau-l] and [lb-tau-r]) or when the two processes exchange a message by proposing complementary actions ([lb-sync]). As usual, we let $\Rightarrow$ stand for the reflexive, transitive closure of $\xrightarrow{\tau}$ and we write $S \mid T \not\rightarrow$ if there are no S' and T' such that $S \mid T \xrightarrow{\tau} S' \mid T'$. Note the different behaviors described by session types of the form $\sum_{i\in I} \pi\mathsf{m}_i.S_i$ depending on the polarity π. According to [lb-tag], a process using a channel of type $\sum_{i\in I} ?\mathsf{m}_i.S_i$ performs an observable transition for each of the tags m_i it is willing to receive. On the contrary, a process using a channel of type $\sum_{i\in I} !\mathsf{m}_i.S_i$ can first *choose* a particular tag $\mathsf{m} = \mathsf{m}_k$ for some $k \in I$ by [lb-pick] (this choice is internal to the process and is therefore unobservable) and then *send* the tag m. As an example, the chain of transitions

$$!\mathsf{m}.S + T \xrightarrow{\tau} !\mathsf{m}.S \xrightarrow{!\mathsf{m}} S$$

models a process that first chooses and then sends the tag m. The choice of the tag is irrevocable and not negotiable with the receiver process. Note that, according to the definition of $+$, T must be an internal choice of tags different from m, hence $!\mathsf{m}.S + T$ is a non-trivial choice among two or more tags.

1.3.2 Multiparty session types

𝄢 A *multiparty* session type describes the communication protocol between at least two participants. For this reason, the types associated to each endpoint are exactly those presented in Section 1.3.1 with the addition of *roles* (p,q $\ldots$). Indeed, a binary session can be considered as the simplest multiparty one with only two participants; hence, the roles can be omitted in the types.

Definition 1.3.2 (Local Types)**.** A *local session type* is a regular tree Courcelle [1983] coinductively generated by the productions

$$S, T, U, V ::= \pi\mathsf{end} \mid \sum_{i\in I} \mathsf{p}\pi\mathsf{m}_i.S_i \mid \mathsf{p}\pi S.T$$

The session type $\pi\mathsf{end}$ describes the behavior of a process that sends or receives a termination signal. The session type $\sum_{i\in I} \mathsf{p}\pi\mathsf{m}_i.S_i$ describes the behavior of a process that sends to or receives from the participant p one of the tags m_i and then behaves according to S_i. Note that the source or destination role p and the polarity π are the same in every branch. We require that I is not empty and $i, j \in I$ with $i \neq j$ implies $\mathsf{m}_i \neq \mathsf{m}_j$. Occasionally we write $\mathsf{p}\pi\mathsf{m}_1.S_1 + \cdots + \mathsf{p}\pi\mathsf{m}_n.S_n$ instead of $\sum_{i=1}^{n} \mathsf{p}\pi\mathsf{m}_i.S_i$. Finally, a session type $\mathsf{p}\pi S.T$ describes the behavior of a process that sends to or receives from the participant p an endpoint of type S and then behaves according to T. We often specify infinite session types as solutions of equations of the form $S = \cdots$ where the metavariable S may occur on the right hand side of $=$ guarded by at least one prefix. A regular tree satisfying such equation is guaranteed to exist and to be unique Courcelle [1983].

In order to describe a whole multiparty session at the level of types we introduce the notion of *session map*.

Definition 1.3.3 (session map)**.** A *session map* is a finite, partial map from roles to session types written $\{\mathsf{p}_i \triangleright S_i\}_{i\in I}$. We let M and N range over session maps, we write $\mathsf{dom}(M)$ for the domain of M, we write $M \mid N$ for the union of M and N when $\mathsf{dom}(M) \cap \mathsf{dom}(N) = \emptyset$, and we abbreviate the singleton map $\{\mathsf{p} \triangleright S\}$ as $\mathsf{p} \triangleright S$.

$$\textsc{lm-end}\quad \frac{}{\mathsf{p} \triangleright \pi\mathsf{end} \xrightarrow{\pi\checkmark} \mathsf{p} \triangleright \pi\mathsf{end}} \qquad \textsc{lm-channel}\quad \frac{}{\mathsf{p} \triangleright \mathsf{q}\pi U.S \xrightarrow{\mathsf{p}\triangleright\mathsf{q}\pi U} \mathsf{p} \triangleright S}$$

$$\textsc{lm-pick}\quad \frac{}{\mathsf{p} \triangleright \sum_{i\in I} \mathsf{q}!\mathsf{m}_i.S_i \xrightarrow{\tau} \mathsf{p} \triangleright \mathsf{q}!\mathsf{m}_k.S_k}\ k \in I$$

$$\textsc{lm-tag}\quad \frac{}{\mathsf{p} \triangleright \sum_{i\in I} \mathsf{q}\pi\mathsf{m}_i.S_i \xrightarrow{\mathsf{p}\triangleright\mathsf{q}\pi\mathsf{m}_k} \mathsf{p} \triangleright S_k}\ k \in I \qquad \textsc{lm-tau}\quad \frac{M \xrightarrow{\tau} M'}{M \mid N \xrightarrow{\tau} M' \mid N}$$

$$\textsc{lm-terminate}\quad \frac{M \xrightarrow{?\checkmark} M' \qquad N \xrightarrow{!\checkmark} N'}{M \mid N \xrightarrow{?\checkmark} M' \mid N'} \qquad \textsc{lm-sync}\quad \frac{M \xrightarrow{\overline{\alpha}} M' \qquad N \xrightarrow{\alpha} N'}{M \mid N \xrightarrow{\tau} M' \mid N'}$$

Figure 1.2: Labeled Transition System for Session Maps

Again, we describe the evolution of a session at the level of types by means of a *labeled transition system* for session maps. Labels are generated by the grammar below:

Label $\quad \ell ::= \tau \mid \alpha \qquad$ **Action** $\quad \alpha, \beta ::= \pi\checkmark \mid \mathsf{p} \triangleright \mathsf{q}\pi\mathsf{m} \mid \mathsf{p} \triangleright \mathsf{q}\pi S$

The label τ represents either an internal action performed by a participant independently of the others or a synchronization between two participants. The labels of the form $\pi\checkmark$ describe the input/output of termination signals, whereas the labels of the form $\mathsf{p} \triangleright \mathsf{q}\pi\mathsf{m}$ and $\mathsf{p} \triangleright \mathsf{q}\pi S$ represent the input/output of a tag m or of an endpoint of type S.

The labeled transition system is defined by the rules in Figure 1.2, most of which are straightforward. Rule [lm-pick] models the fact that the participant p may internally choose one particular tag m_k before sending it to q. The chosen tag is not negotiable with the receiver. Rule [lm-terminate] models termination of a session. A session terminates when there is exactly one participant waiting for the termination signal and all the others are sending it. This property follows from a straightforward induction on the derivation of $M \xrightarrow{?\checkmark} N$ using [lm-terminate] and [lm-end].

The existence of a single participant waiting for the termination signal ensures that there is a uniquely determined continuation process after the session has been closed. Finally, rule [lm-sync] models the synchronization between two participants performing complementary actions. The complement of an action α, denoted by $\overline{\alpha}$, is the partial operation defined by the equations

$$\overline{\mathsf{p} \triangleright \mathsf{q}\pi\mathsf{m}} \overset{\mathsf{def}}{=} \mathsf{q} \triangleright \mathsf{p}\overline{\pi}\mathsf{m} \qquad \overline{\mathsf{p} \triangleright \mathsf{q}\pi S} \overset{\mathsf{def}}{=} \mathsf{q} \triangleright \mathsf{p}\overline{\pi}S$$

where $\overline{\pi}$ denotes the complement of the polarity π. The complement of actions of the form $\pi\checkmark$ is undefined, so rule [LM-SYNC] cannot be applied to terminated sessions. Hereafter we write $\Rightarrow$ for the reflexive, transitive closure of $\xrightarrow{\tau}$ and $\overset{\alpha}{\Longrightarrow}$ for the composition $\Rightarrow\xrightarrow{\alpha}$.

1.4 Related Work

Now we have all the ingredients for discussing about some well known properties that can be found in the literature. Notably, such properties have been studied both on the pure π-calculus and on session-based calculi. We split the discussion according the property and the scenario under analysis.

Termination of Binary Sessions. Termination is a liveness property that can be guaranteed when finite session types are considered [Pérez et al., 2012]. As soon as infinite session types are considered, many session type systems weaken the guaranteed property to deadlock freedom. Lindley and Morris [2016] define a type system for a functional language with session primitives and recursive session types that is strongly normalizing.

Liveness Properties in the π-Calculus. Kobayashi [2002a] defines a behavioral type system that guarantees lock freedom in the π-calculus. Lock freedom is a liveness property akin to progress for sessions, except that it applies to *any* communication channel (shared or private). As a paradigmatic example of lock we can consider a variant of Example 2.1.1 in which the **buyer** never pays the products. Thus, the **carrier** is locked. Padovani [2014] adapts and extends the type system of Kobayashi [2002a] to enforce lock freedom in the *linear π-calculus* [Kobayashi et al., 1999], into which binary sessions can be encoded [Dardha et al., 2017]. All of these works annotate types with numbers representing finite upper bounds to the number of interactions needed to unblock a particular input/output action.

Deadlock Freedom. Deadlock freedom is a safety properties according to which a communication cannot get stuck. A paradigmatic example of deadlocked process is (we adopt binary session type syntax)

$$x?\mathsf{true}.y!\mathsf{false}\cdots \mid y?\mathsf{false}.x!\mathsf{true}\ldots$$

where we have a couple of processes running in parallel. The former is waiting for true on x which is sent by the second as soon as it receives false by the former.

Kobayashi [2002a], Padovani [2014] guarantee deadlock freedom using the same technique for lock freedom. There exist a number of session based type systems inspired by linear logic [Wadler, 2014, Caires et al., 2016, Lindley and Morris, 2016, Carbone et al., 2016, 2017] in which deadlock freedom is dealt with by using a *process composition* rule that resembles the *cut* rule.

Liveness Properties of Multiparty Sessions. The enforcement of liveness properties has always been a key aspect of session type systems, although previous works have almost exclusively focused on progress rather than on (fair) termination. Scalas and Yoshida [2019] define a general framework for ensuring safety and liveness properties of multiparty sessions. In particular, they define a hierarchy of three liveness predicates to characterize "live" sessions that enjoy progress. They also point out that the coarsest liveness property in this hierarchy, which is the one more closely related to fair termination, cannot be enforced by their type system. The work of van Glabbeek et al. [2021] presents a type system for multiparty sessions that ensures progress and is not only sound but also complete. Carbone et al. [2014] characterize progress in terms of the standard notion of lock-freedom by using catalysers.

Chapter 2

Fair Termination

Among the properties mixing safety and liveness aspects in the context of session types, we decided to develop type systems enforcing *fair termination*. Such property is desirable for many reasons. Indeed, as we briefly mentioned at the beginning of the thesis, a lock free session does not imply that other sessions are lock free as well. On the other hand, if we consider a session and we assume that all the other ones are fairly terminating, we can conclude that the one under analysis is fairly terminating as well.

In Section 2.1 we describe the property and we relate it with other properties that are frequently studied in the literature. In Section 2.2 we formally define *fair termination* in its general form on *reduction systems* such that it will be instantiated in the different session type based scenarios in Section 2.3. Notably, in Part III we will provide a characterization of fair termination based on generalized inference systems (Section 1.2) and we will provide a mechanization of both its definition and correctness proofs in Agda.

2.1 Introduction

The decomposition of a distributed program into sessions enables its modular static analysis and the enforcement of useful properties through a type system. Examples of such properties are *communication safety* (no message of the wrong type is ever exchanged), *protocol fidelity* (messages are exchanged in the order prescribed by session types) and *deadlock freedom* (the program keeps running unless all sessions have terminated). These are all instances of *safety properties*, implying that “nothing bad” happens. In general, one is also interested in reasoning and possibly enforcing *liveness properties*, those implying that “something good” happens Owicki and Lamport [1982]. Examples of liveness properties are *junk freedom* (every message is eventually received), *progress* (every non-terminated participant of multiparty a session eventually performs an action) and *termination* (every session eventually comes to an end).

2.1.1 What

𝄢 Fair termination is *termination* under a *fairness assumption*. We decided to investigate such property since current type systems cannot deal with those scenarios in which the communication between two entities *depends* on the communication between others. To explain why, we show a paradigmatic scenario that we will instantiate in different ways in the next chapters.

Example 2.1.1 (Buyer - Seller - Carrier)**.**
Consider the interaction between the following three entities:

- ***buyer**: he can tell a **seller** either that he adds an item to the cart or that he pays the total amount. We assume that the messages being sent are* add *and* pay*, respectively*

- ***seller**: if the **buyer** decides to pay the amount (*i.e. pay *is received) he contacts the **carrier** to ship the items. The message being sent is* ship

- ***carrier**: he sends the items as soon as he is contacted by the **seller** (*i.e. ship *is received)*

At the moment we do not care about the technicalities on how the three actors interact. What makes this scenario somewhat difficult to reason about is that *the progress of the carrier is not unconditional but depends on the choices performed by the buyer*: the carrier can make progress only if the buyer eventually pays the seller.

The *fairness assumption* that we used can be informally stated as

If termination is always possible, then it is inevitable

Note that Example 2.1.1 admits an infinite execution in which the **buyer** only adds items to the cart and never pays the amount. Such execution is *unfair* according to our assumption since the **buyer** can always send pay (and terminate) but he always avoid to do so.

2.1.2 Why

The reader might wonder why we focus on fair termination instead of considering some fair version of progress. There are three reasons why we think that fair termination is overall more appropriate than just progress. First of all, ensuring that sessions (fairly) terminate is consistent with the usual interpretation of the word "session" as an activity that lasts for a *finite amount of time*, even when the maximum duration of the activity is not known *a priori*. Second, *fair termination implies progress* when it is guaranteed along with the usual safety properties of sessions. Indeed, if the session eventually terminates, it must be the case that any non-terminated participant (think of the carrier waiting for a *"ship"* message) is guaranteed to eventually make progress, even when such progress *depends* on choices made by others (like the buyer sending *"pay"* to the seller). Last but not least, *fair session termination enables compositional reasoning* in the presence of multiple sessions. This is not true for progress: if an action on a session s is blocked by actions on a different session t, then knowing that the session t enjoys progress does not necessarily guarantee that the action on s will eventually be performed (the interaction on t might continue forever). On the contrary, knowing that t fairly terminates guarantees that the action on s will eventually be scheduled and performed, so that s may in turn progress towards termination.

2.2 Fair Termination - Formally

Since the notion of fair termination will apply to several different entities in this chapter (session types, binary and multiparty sessions, processes) here we formally define it for a generic reduction system. Later on we will show various instantiations of this definition.

Definition 2.2.1 (Reduction System)**.** A *reduction system* is a pair $(\mathcal{S}, \to)$ where

- $\mathcal{S}$ is a set of *states*

- $\rightarrow \subseteq \mathcal{S} \times \mathcal{S}$ is a *reduction relation*

We adopt the following notation: we let C and D range over states; we write $C \rightarrow$ if there exists $D \in \mathcal{S}$ such that $C \rightarrow D$; we write $C \nrightarrow$ if not $C \rightarrow$; we write $\Rightarrow$ for the reflexive, transitive closure of $\rightarrow$. We say that D is *reachable* from C if $C \Rightarrow D$.

As an example, the reduction system $(\{A, B\}, \{(A, A), (A, B)\})$ models an entity that can be in two states, A or B, and such that the entity may perform a reduction to remain in state A or a reduction to move from state A to state B. To formalize the evolution of an entity from a particular state we define *runs*.

Definition 2.2.2 (runs and maximal runs)**.** A *run* of C is a (finite or infinite) sequence $C_0C_1 \dots C_i \dots$ of states such that $C_0 = C$ and $C_i \rightarrow C_{i+1}$ for every valid i. A run is *maximal* if either it is infinite or if its last state C_n is such that $C_n \nrightarrow$.

Hereafter we let ρ range over runs. Each run in the previously defined reduction system is either of the form A^n – a finite sequence of A – or of the form A^nB – a finite sequence of A followed by one B – or A^ω – an infinite sequence of A. Among these, the runs of the form A^nB and A^ω are maximal, whereas no run of the form A^n is maximal.

We now use runs to define different termination properties of states: we say that C is *weakly terminating* if there exists a maximal run of C that is finite; we say that C is *terminating* if every maximal run of C is finite; we say that C is *diverging* if every maximal run of C is infinite. *Fair termination* [Francez, 1986] is a termination property that only considers a subset of all (maximal) runs of a state, those that are considered to be "realistic" or "fair" according to some fairness assumption. The assumption that we make in this work, and that we stated in words in Section 2.1, is formalized thus:

Definition 2.2.3 (Fair run)**.** A run is *fair* if it contains finitely many weakly terminating states. Conversely, a run is *unfair* if it contains infinitely many weakly terminating states.

Continuing with the previous example, the runs of the form A^n and A^nB are fair, whereas the run A^ω is unfair. In general, an unfair run is an execution in which termination is always within reach, but is never reached.

A key requirement of any fairness assumption is that it must be possible to extend every finite run to a maximal fair one. This property is called *feasibility* [Apt et al., 1987, van Glabbeek and Höfner, 2019] or *machine* *closure* [Lamport, 2000]. It is easy to see that our fairness assumption is feasible:

Lemma 2.2.1. If ρ is a finite run, then there exists ρ' such that $\rho\rho'$ is a maximal fair run.

Proof. Let D be the last state of ρ. We distinguish two possibilities: if D is weakly terminating, then there exists a finite maximal run $D\rho'$ of D; if D is diverging, then there exists an infinite run $D\rho'$ of D such that no state in ρ' is weakly terminating. In both cases we conclude by noting that $\rho\rho'$ is a maximal fair run. □

Fair termination is finiteness of all maximal fair runs:

Definition 2.2.4 (Fair Termination)**.** We say that C is *fairly terminating* if every maximal fair run of C is finite.

In the reduction system given above, A is fairly terminating. Indeed, all the maximal runs of the form $A^n B$ are finite whereas A^ω, which is the only infinite run of A, is unfair.

For the particular fairness assumption that we make, it is possible to provide a sound and complete characterization of fair termination that does not mention fair runs. This characterization will be useful to relate fair termination with the notion of correct session (Definitions 2.3.1 and 2.3.2) and the soundness property of the type systems we are going to introduce in Part II.

Theorem 2.2.1. *Let $(\mathcal{S}, \rightarrow)$ be a reduction system and $C \in \mathcal{S}$. Then C is fairly terminating if and only if every state reachable from C is weakly terminating.*

Proof. $\Rightarrow$. Let D be a state reachable from C. That is, there exists a finite run ρ of C ending with D. By Lemma 2.2.1 we deduce that this run can be extended to a maximal fair one $\rho\rho'$. From the hypothesis that C is fairly terminating we deduce that $\rho\rho'$ is finite. Hence, D is weakly terminating.

$\Leftarrow$. Let $C_0C_1\dots$ be an infinite fair run of C. Using the hypothesis we deduce that each C_i is weakly terminating, which is absurd. Hence, either there are no maximal fair runs or every maximal fair run of C is finite, but the first case is not possible by Lemma 2.2.1, thus C is fairly terminating. □

Remark 2.2.1 (Fair Reachability of Predicates)*.* Most fairness assumptions have the form "if *something* is infinitely often possible then *something* happens infinitely often" and, in this respect, our formulation of fair run (definition 2.2.3) looks slightly unconventional. However, it is not difficult to realize that definition 2.2.3 is an instance of the notion of fair reachability of predicates as defined by [Queille and Sifakis, 1983, Definition 3]. According to Queille and Sifakis, a run ρ is fair with respect to some predicate $\mathcal{C} \subseteq \mathcal{S}$ if, whenever in ρ there are infinitely many states from which a state in $\mathcal{C}$ is reachable, then in ρ there are infinitely many occurrences of states in $\mathcal{C}$. When we take $\mathcal{C}$ to be $\not\rightarrow$, that is the set of terminated states that do not reduce, pretending that irreducible states should occur infinitely often in the run is nonsensical. So, the fairness assumption boils down to assuming that such states should *not* be reachable infinitely often, which is precisely the formulation of definition 2.2.3. ⌟

2.3 Fair Termination and Session Types

In this section we instantiate *fair termination* in session type based scenarios. First, we consider binary sessions and then we generalize to the multiparty case. We instantiate Example 2.1.1 as well since it will be the running example in Chapter 4 and Chapter 5. For what concerns the used syntaxes and labeled transition systems, we refer to Section 1.3.

Definition 2.3.1 (Compatibility of a binary session)**.** We say that S and T are *compatible*, notation $S \sim T$, if $S \mid T \Rightarrow S' \mid T'$ implies $S' \mid T' \Rightarrow \overline{\pi}\mathsf{end} \mid \pi\mathsf{end}$ for some π.

Definition 2.3.2 (Coherence of a session map)**.** We say that a session map M is *coherent*, notation $\#M$, if $M \Rightarrow N$ implies $N \xRightarrow{?\checkmark}$.

The term "coherence" is borrowed from Carbone et al. [2016, 2017], although the property is actually stronger than the one of Carbone et al. [2016, 2017] as it entails fair termination of multiparty sessions through theorem 2.2.1. In particular, if we consider the reduction system whose states are session maps and whose reduction relation is $\xrightarrow{\tau}$, then $\#M$ implies M fairly terminating. The same applies to a $S \sim T$ binary session.

Remark 2.3.1 (Successful fair termination). The properties stated in Definitions 2.3.1 and 2.3.2 are stronger than fair termination. Indeed, a *deadlocked* session is *fairly terminating* as well as it cannot reduce. Definitions 2.3.1 and 2.3.2 are equivalent to a *successful* form of fair termination since we ask that the involved sessions correctly terminate.

Now we can revise Example 2.1.1 in both scenarios.

Example 2.3.1 (Binary Buyer - Seller - Carrier)**.** *Consider the types S_b, S_s and T_s, T_c that model the two binary sessions connecting* ***buyer*** *-* ***seller*** *and* ***seller*** *-* ***carrier****, respectively, according to Example 2.1.1.*

$$\begin{array}{rclcrcl} S_b & = & !\mathsf{add}.S_b + !\mathsf{pay}.!\mathsf{end} & \quad & T_s & = & !\mathsf{ship}.!\mathsf{end} \\ S_s & = & ?\mathsf{add}.S_s + ?\mathsf{pay}.?\mathsf{end} & & T_c & = & ?\mathsf{ship}.?\mathsf{end} \end{array}$$

We focus on the session $S_b \mid S_s$. According to Definition 2.3.1, S_b and S_s are compatible *because, no matter how many times the* ***buyer*** *adds an item to the cart, he always has the possibility to* pay *the amount. The infinite run in which the* ***buyer*** *only adds items is* unfair *according to Definition 2.2.3.*

Note that $S_b \sim S_s$ implies progress *of the session $T_s \mid T_c$. Indeed, the communication between the* ***seller*** *and the* ***carrier*** *takes place after the* ***buyer*** *sends* pay *which is guaranteed to happen by $S_b \sim S_s$. Furthermore, although this example can be easily adapted to the multiparty case, it shows a very simple and realistic scenario in which more sessions are involved, no matter if they are binary or multiparty.* ⌟

Example 2.3.1 clearly holds in the multiparty context as well, now we can model the same communication protocol using a single multiparty session.

Example 2.3.2 (Multiparty Buyer - Seller - Carrier)**.** *Consider the session types*

$$\begin{array}{rcl} S_b & = & \mathsf{seller}!\mathsf{add}.S_b + \mathsf{seller}!\mathsf{pay}.!\mathsf{end} \\ S_s & = & \mathsf{buyer}?\mathsf{add}.S_s + \mathsf{buyer}?\mathsf{pay}.\mathsf{carrier}!\mathsf{ship}.!\mathsf{end} \\ S_c & = & \mathsf{seller}?\mathsf{ship}.?\mathsf{end} \end{array}$$

which describe the behavior of the three participants in example 2.1.1. The session map $\mathsf{buyer} \triangleright S_b \mid \mathsf{seller} \triangleright S_s \mid \mathsf{carrier} \triangleright S_c$ is coherent*. The explanation is just the same as the one given in Example 2.3.1.* ⌟

Notably, in the multiparty context, fair termination implies *progress* of all non terminated participants (see carrier in Example 2.3.2).

Chapter 3

Fair Subtyping

The mere *assumption* of fairness (see Definition 2.2.3) does not turn an ordinary session type system into one that ensures fair session termination because the correspondence imposed by the type system between the structure of processes and that of the protocols they implement is generally (often necessarily) a loose one. Indeed, processes may be "more accommodating" than the protocols they implement by handling more messages than those mentioned in the protocols. For example, the **seller** in Example 2.1.1 could handle a search message in addition to add and pay, even if the session type associated with x does not mention search. At the same time, processes may also be "less demanding" than the protocols they implement by sending fewer messages than those allowed by the protocols. For example, the **buyer** in Example 2.1.1 could always purchase an even/odd number of items, or at least n items, or no more than n items, even if the session type associated with the channel allows sending an arbitrary number of add messages. These mismatches between processes and protocols are usually reconciled by a *subtyping relation* for session types [Gay and Hole, 2005, Bernardi and Hennessy, 2016]. The problem is that this subtyping relation is *too coarse* because it has been conceived to preserve the *safety* properties of sessions but not termination, which is a *liveness* property: if session types are not sufficiently precise descriptions of the actual behavior of processes, a session that appears to be fairly terminating at the level of types may not terminate at all at the level of processes. To solve this problem we adopt *fair subtyping* [Padovani, 2013, 2016, Bravetti et al., 2021], a *liveness-preserving* refinement of the subtyping relation defined by Gay and

$$\textsc{us-end}\ \frac{}{\pi\mathsf{end} \leqslant_* \pi\mathsf{end}} \qquad \textsc{us-channel-in}\ \frac{S \leqslant_* T \quad U \leqslant_* V}{?U.S \leqslant_* ?V.T} \qquad \textsc{us-channel-out}\ \frac{S \leqslant_* T \quad V \leqslant_* U}{!U.S \leqslant_* !V.T}$$

$$\textsc{us-tag-in}\ \frac{\forall i \in I : S_i \leqslant_* T_i}{\sum_{i\in I} ?\mathsf{m}_i.S_i \leqslant_* \sum_{i\in I\cup J} ?\mathsf{m}_i.T_i} \qquad \textsc{us-tag-out}\ \frac{\forall i \in I : S_i \leqslant_* T_i}{\sum_{i\in I\cup J} !\mathsf{m}_i.S_i \leqslant_* \sum_{i\in I} !\mathsf{m}_i.T_i}$$

Figure 3.1: Rules for subtyping [Gay and Hole, 2005]

Hole [2005].

The chapter is organized as follows. First, in Section 3.1 we present the original subtyping relation for session types [Gay and Hole, 2005]. We conclude such section showing why such relation is *unfair* by applying it to a variant of Example 2.1.1 (see Section 3.1.1). Then, in Section 3.2 we introduce *fair subtyping* [Padovani, 2013, 2016, Bravetti et al., 2021] . We show an inference system for characterizing it and we prove its correctness with respect to a semantic definition. At last, we show a characterization of fair subtyping by using a generalized inference system (see Section 3.2.3). We will refer to this definition in Section 8.5.

3.1 Original Subtyping

The original subtyping relation for session types has been introduced by Gay and Hole [2005] and it is obtained by coinductively interpreting the rules in Figure 3.1. Notably, we only show the relation for binary session types since in the multiparty case it is defined in the same way and it can be obtained by simply adapting the syntax. The inference system in Figure 3.1 derives judgments of the form $S \leqslant_* T$ meaning that S is a *subtype* of T. Let us analyze the rules in details.

Rule [US-END] is used to relate terminated sessions. [US-CHANNEL-IN] and [US-CHANNEL-OUT] relate input and output of channels, respectively. Note that they show a different behavior. While the former is *covariant* with respect to both the channel being received and the continuation, the second one is *contravariant* in the exchanged channel. Rules [US-TAG-IN] and [US-TAG-OUT] relate input and output of message tags, respectively. Similarly to the exchange of a channel, the former is covariant while the second contravariant in the set of messages being exchanged.

The rules are consistent with the informal example that we gave at the beginning of Chapter 3 by referring to Example 2.1.1. Indeed, the **seller** in could handle a search message in addition to add and pay while the **buyer** could always purchase an even/odd number of items.

Example 3.1.1. *Consider the session types S_b and S_s from Example 2.3.1 describing the communication protocol between the* ***buyer*** *and the* ***seller****. We can derive*

$$\begin{array}{lcl} S_b = !\mathsf{add}.S_b + !\mathsf{pay}.!\mathsf{end} & \leqslant_* & S'_b = !\mathsf{add}!\mathsf{add}.S'_b + !\mathsf{pay}.!\mathsf{end} \\ S_s = ?\mathsf{add}.S_s + ?\mathsf{pay}.?\mathsf{end} & \leqslant_* & S'_s = ?\mathsf{add}.S'_s + ?\mathsf{pay}.?\mathsf{end} + ?\mathsf{search}.S'_s \end{array}$$

We can focus on the infinite derivation trees.

$$\dfrac{\dfrac{\dfrac{\vdots}{S_b \leqslant_* S'_b}}{!\mathsf{add}.S_b + !\mathsf{pay}.!\mathsf{end} \leqslant_* !\mathsf{add}.S'_b}\,[\text{US-TAG-OUT}] \qquad \dfrac{}{!\mathsf{end} \leqslant_* !\mathsf{end}}\,[\text{US-END}]}{!\mathsf{add}.S_b + !\mathsf{pay}.!\mathsf{end} \leqslant_* !\mathsf{add}!\mathsf{add}.S'_b + !\mathsf{pay}.!\mathsf{end}}\,[\text{US-TAG-OUT}]$$

$$\dfrac{\dfrac{\vdots}{S_s \leqslant_* S'_s} \qquad \dfrac{}{?\mathsf{end} \leqslant_* ?\mathsf{end}}\,[\text{US-END}]}{?\mathsf{add}.S_s + ?\mathsf{pay}.?\mathsf{end} \leqslant_* ?\mathsf{add}.S'_s + ?\mathsf{pay}.?\mathsf{end} + ?\mathsf{search}.S'_s}\,[\text{US-TAG-IN}]$$

As previously noted, the same judgments can be derived for the multiparty session types in Example 2.3.2. ⌟

The subtyping relation we presented induces a substitution principle.

Proposition 3.1.1 (*Safe* substitution principle)**.** *If $S \leqslant_* T$, then a process that uses an endpoint according to S can be* safely *substituted with a process that uses the endpoint according to T.*

Note that we highlight the word *safe* as we want to point out that the subtyping relation under analysis is studied to preserve the safety of the communication. We will give more details in Section 3.1.1.

Remark 3.1.1. The reader might be confused by the formulation of Proposition 3.1.1 as it seems to treat the substitution in the wrong direction (left-to-right). However, it is important to think about the subject of the principle, *i.e.* processes in Proposition 3.1.1. Indeed, the same principle can be formulated in a right-to-left fashion as in Liskov and Wing [1994] by taking into account the endpoints and not the processes. The two formulations turn out to be equivalent (see Gay [2016] for more details). ⌟

3.1.1 Unfair Subtyping

In Section 3.1 we pointed out the *safety*-preserving property of Proposition 3.1.1. Indeed, the two substitutions proposed in Example 3.1.1 do not break the correctness of the communication between the entities. However, the subtyping relation introduced by Gay and Hole [2005] can break the *liveness* of the session on which it is applied.

Example 3.1.2. *Consider the session types S_b and S_s from Example 2.3.1. The following judgment can be derived*

$$S_b \quad = \quad !\mathsf{add}.S_b + !\mathsf{pay}.!\mathsf{end} \quad \leqslant_* \quad S_b^\infty \quad = \quad !\mathsf{add}.S_b^\infty$$

Let us look at the infinite derivation tree

$$\dfrac{\dfrac{\dfrac{\vdots}{S_b \leqslant_* S_b^\infty}\,[\textsc{us-tag-out}]}{S_b \leqslant_* S_b^\infty}\,[\textsc{us-tag-out}]}{!\mathsf{add}.S_b + !\mathsf{pay}.!\mathsf{end} \leqslant_* !\mathsf{add}.S_b^\infty}\,[\textsc{us-tag-out}]$$

As a consequence, ***buyer*** *can be replaced with* ***buyer***$_\infty$ *behaving as* S_b^∞ *by using Proposition 3.1.1 (it only* add*s items). No matter how it reduces, the* ***buyer***$_\infty$ *will never* pay *the amount and the session will never terminate. Moreover, the only run of the session is infinite and* fair *since it does not contain any weakly terminating state. This example applies to Example 5.1.1 as well.* ⌟

Intuitively, the problem of the subtyping we investigated so far lies in the contravariance of the output rule [US-TAG-OUT]. When such rule is applied, some branches are cut and it might happen that all those branches leading to termination are removed. This is what happens in Example 3.1.2. Hence, the subtyping of Gay and Hole [2005] must be refined to preserve the termination property of the session.

3.2 Fair Subtyping

Fair subtyping is a *liveness*-preserving variant of the subtyping proposed by Gay and Hole [2005]. In this section we present such relation as a purely coinductive predicate. However, in Section 3.2.3 we will show an

$$\textsc{fsb-end}\ \frac{}{\pi\mathsf{end} \leqslant_n \pi\mathsf{end}} \qquad \textsc{fsb-tag-in}\ \frac{\forall i \in I : S_i \leqslant_{n_i} T_i \qquad \forall i \in I : n_i \leq n}{\sum_{i\in I} ?\mathsf{m}_i.S_i \leqslant_n \sum_{i\in I\cup J} ?\mathsf{m}_i.T_i}$$

$$\textsc{fsb-channel}\ \frac{S \leqslant_n T}{\pi U.S \leqslant_n \pi U.T} \qquad \textsc{fsb-tag-out-1}\ \frac{\forall i \in I : S_i \leqslant_{n_i} T_i \qquad \forall i \in I : n_i \leq n}{\sum_{i\in I} !\mathsf{m}_i.S_i \leqslant_n \sum_{i\in I} !\mathsf{m}_i.T_i}$$

$$\textsc{fsb-tag-out-2}\ \frac{\forall i \in I : S_i \leqslant_{n_i} T_i \qquad \exists i \in I : n_i < n}{\sum_{i\in I\cup J} !\mathsf{m}_i.S_i \leqslant_n \sum_{i\in I} !\mathsf{m}_i.T_i}$$

Figure 3.2: Inference system for fair subtyping - Binary

$$\textsc{fsm-end}\ \frac{}{\pi\mathsf{end} \leqslant_n \pi\mathsf{end}} \qquad \textsc{fsm-tag-in}\ \frac{\forall i \in I : S_i \leqslant_{n_i} T_i \qquad \forall i \in I : n_i \leq n}{\sum_{i\in I} \mathsf{p}?\mathsf{m}_i.S_i \leqslant_n \sum_{i\in I\cup J} \mathsf{p}?\mathsf{m}_i.T_i}$$

$$\textsc{fsm-channel}\ \frac{S \leqslant_n T}{\mathsf{p}\pi U.S \leqslant_n \mathsf{p}\pi U.T} \qquad \textsc{fsm-tag-out-1}\ \frac{\forall i \in I : S_i \leqslant_{n_i} T_i \qquad \forall i \in I : n_i \leq n}{\sum_{i\in I} \mathsf{p}!\mathsf{m}_i.S_i \leqslant_n \sum_{i\in I} \mathsf{p}!\mathsf{m}_i.T_i}$$

$$\textsc{fsm-tag-out-2}\ \frac{\forall i \in I : S_i \leqslant_{n_i} T_i \qquad \exists i \in I : n_i < n}{\sum_{i\in I\cup J} \mathsf{p}!\mathsf{m}_i.S_i \leqslant_n \sum_{i\in I} \mathsf{p}!\mathsf{m}_i.T_i}$$

Figure 3.3: Inference system for fair subtyping - Multiparty

equivalent and more involved formulation obtained by equipping the rules in Figure 3.1 with a *corule* (see Section 1.2). While the generalized inference system is intriguing since it points out the safety and liveness aspects of the predicate, the formulation we give in this section significantly simplifies the correctness proofs and it will be easily integrated in the type systems in Part II. Since binary sessions are the simplest multiparty ones, we refer to the binary case when it allows to simplify the presentation of some features of the property (*e.g.* examples, description of the rules) while we present all the results in the more general multiparty one.

Fair subtyping for binary session types is defined as the relation $\leqslant_n$ coinductively defined by the inference system in Figure 3.2, where n ranges over natural numbers. The characterization of fair subtyping that we consider is the relation $\leqslant \overset{\mathsf{def}}{=} \bigcup_{n\in\mathbb{N}} \leqslant_n$. Fair subtyping for multiparty session types is analogously defined (see Figure 3.3). The rules for deriving $S \leqslant_n T$ are quite similar to those of the standard subtyping relation for session types [Gay and Hole, 2005]: [FSB-END] states reflexivity of subtyping on terminated session types; [FSB-CHANNEL] relates higher-order session types with the same polarity and payload type; [FSB-TAG-IN] is the usual covariant rule for the input of tags (the set of tags in the larger session type includes those in the smaller one); [FSB-TAG-OUT-2] is the usual contravariant rule for the output of tags (the set of tags in the smaller session type includes those in the larger one). Overall, these rules entail a "simulation" between the behaviors described by S_b and S'_b whereby all inputs offered by S_b are also offered by S'_b and all outputs performed by S'_b are also performed by S_b. The main differences between $\leqslant$ and the subtyping relation of Gay and Hole [2005] are the presence of an invariant rule for outputs [FSB-TAG-OUT-1] and the natural number n annotating each subtyping judgment $S \leqslant_n T$. Intuitively, this number estimates how much S_b and S'_b differ in terms of performed outputs. In all rules but [FSB-TAG-OUT-2], the annotation in the conclusion of the rule is just an upper bound of the annotations found in the premises. In [FSB-TAG-OUT-2], where the sets of output tags in related session types may differ, the annotation n is required to be a *strict* upper bound for at least one of the premises. That is, there must be at least one premise in which the annotation strictly decreases, while no restriction is imposed on the others. Intuitively, this ensures the existence of a tag shared by the two related session types whose corresponding continuations are slightly less different. So, the annotation n provides an upper bound to the number of applications of [FSB-TAG-OUT-2] along any path (*i.e.* any sequence of actions) shared by S_b and S'_b that leads to termination. In the particular case when $n = 0$, the rule [FSB-TAG-OUT-2] cannot be applied, so that S'_b may perform all the outputs also performed by S_b.

Remark 3.2.1 (Invariant delegation). As it can be noted from [FSB-CHANNEL] and [FSM-CHANNEL], we require that the type of the channel being exchanged in the supertype matches that in the subtype. We made such a decision since we found out that allowing co/contravariance of input/output of channels may break the liveness of the session. We will give more details in Section 4.2.4. ⌟

Fair subtyping allows us to reject the subtyping instance in Example 3.1.2 that breaks the liveness of the involved session.

Example 3.2.1. *Consider the session types*

$$S_b = !\mathsf{add}.S_b + !\mathsf{pay}.!\mathsf{end} \qquad S'_b = !\mathsf{add}.!\mathsf{add}.S'_b + !\mathsf{pay}.!\mathsf{end}$$

from Examples 2.3.1 and 3.1.1 describing the ***buyer*** *purchasing an arbitrary number of items and the behavior of the* ***buyer*** *always purchasing an even number of items, respectively. Consider also*

$$S_b^\infty = \,!\mathsf{add}.S_b^\infty$$

from Example 3.1.2, which describes the behavior of a ***buyer*** *attempting to* add *an infinite number of items without ever* pay*ing the* ***seller****. We have* $S_b \leqslant S_b'$ *and* $S_b \not\leqslant S_b^\infty$*. Indeed, we can derive*

$$\dfrac{\dfrac{\dfrac{\vdots}{S_b \leqslant_1 S_b'}}{S_b \leqslant_2 \,!\mathsf{add}.S_b'}\,[\textsc{fsb-tag-out-2}] \qquad \dfrac{}{!\mathsf{end} \leqslant_0 \,!\mathsf{end}}\,[\textsc{fsb-end}]}{S_b \leqslant_1 S_b'}\,[\textsc{fsb-tag-out-2}]$$

but there is no derivation for $S_b \leqslant_n S_b^\infty$ *no matter how large* n *is chosen. Note that there are infinitely many sequences of actions of* S_b *that cannot be performed by both* S_b' *and* S_b^∞*. In particular,* S_b' *cannot perform any sequence of actions consisting of an odd number of* add *outputs followed by a* pay *output, whereas* S_b^∞ *cannot perform any sequence of* add *outputs followed by a* pay *output. Nonetheless, there is a path shared by* S_b *and* S_b' *that leads into a region of* S_b *and* S_b' *in which no more differences are detectable. The annotations in the derivation tree measures the distance of each judgment from such region. In the case of* S_b *and* S_b^∞*, there is no shared path that leads to a region where no differences are detectable.* ⌟

Example 3.2.2. *Consider the session types*

$$\begin{array}{rcl} S & = & ?\mathsf{play}.(!\mathsf{win}.S + !\mathsf{lose}.S) + ?\mathsf{quit}.!\mathsf{end} \\ T & = & ?\mathsf{play}.!\mathsf{lose}.T + ?\mathsf{quit}.!\mathsf{end} \end{array}$$

describing the behavior of two slot machines, an unbiased one in which the player may win at every play and a biased one in which the player never wins. If we try to build a derivation for $S \leqslant_n T$ *we obtain*

$$\dfrac{\dfrac{\dfrac{\vdots}{S \leqslant_{n-1} T}}{!\mathsf{win}.S + !\mathsf{lose}.S \leqslant_n \,!\mathsf{lose}.T}\,[\textsc{f-tag-out-1}] \qquad \dfrac{}{!\mathsf{end} \leqslant_n \,!\mathsf{end}}\,[\textsc{f-end}]}{S \leqslant_n T}\,[\textsc{f-tag-in}]$$

which would contain an infinite branch with strictly decreasing annotations. Therefore, we have $S \not\leqslant T$*. In this case there exists a shared path leading* *into a region of S and T in which no more differences are detectable between the two protocols, but this path starts from an input. The fact that S is* not *a fair subtype of T has a semantic justification. Think of a player that deliberately insists on playing until it wins. This is possible when the player interacts with the unbiased slot machine S but not with the biased one T.* ⌟

Now we can provide a semantic characterization of fair subtyping for binary and multiparty session types as the relation preserving *compliance* (Definition 2.3.1) and *coherence* (Definition 2.3.2) of the involved session, respectively. Since the notion of *coherence* boils down to *compatibility* when there are exactly two participants, we only state the property in the multiparty scenario.

Definition 3.2.1 (Semantic fair subtyping - Multiparty)**.** We say that S is a *fair subtype* of T, notation $S \sqsubseteq T$, if $M \mid \mathsf{p} \triangleright S$ *coherent* implies $M \mid \mathsf{p} \triangleright T$ *coherent* for every M and p.

We conclude this section by stating and proving a fundamental property of fair subtyping.

Theorem 3.2.1. $\leqslant$ *is a preorder.*

While reflexivity of $\leqslant$ is trivial to prove (since [FSM-TAG-OUT-2] is never necessary, it suffices to only consider judgments with a 0 annotation), transitivity is surprisingly complex. The challenging part of proving that from $S \leqslant_m U$ and $U \leqslant_n T$ we can derive $S \leqslant_k T$ is to come up with a feasible annotation k. As it turns out, such k depends not only on m and n, but also on annotations found in different regions of the derivation trees that prove $S \leqslant_m U$ and $U \leqslant_n T$. In particular, the "difference" of S and T is not simply the "maximum difference" or "the sum of the differences" of S and U and of U and T. More in detail, we first show that we can always find a derivation of $S \leqslant_m U$ where the rank annotations of all judgements occurring in it are below some $h \geq m$; then, the judgement $S \leqslant_k T$ is provable for $k = m + (1 + h)n$.

Lemma 3.2.1. Let $S \leqslant_n^m T$ if and only if $S \leqslant_n T$ is the conclusion of a derivation in which every rank annotation is at most m. Then $S \leqslant_n T$ if and only $S \leqslant_n^m T$ for some m.

Proof. The "if" part is obvious. Concerning the "only if" part, it suffices to show that each judgment in the set

$$\mathcal{S} \overset{\mathsf{def}}{=} \{U \leqslant_m V \mid U \leqslant_m V \wedge \nexists n < m : U \leqslant_n V\}$$

is derivable from premises that are also in $\mathcal{S}$. This is enough to prove $S \leqslant_n^m T$ from $S \leqslant_n T$, because in $\mathcal{S}$ there is at most one judgment $U \leqslant_n V$ for each pair of session types U and V and, by regularity of U and V, the derivation of $U \leqslant_n V$ obtained using judgments in $\mathcal{S}$ contains finitely many annotations, which must have a maximum.

Suppose $U \leqslant_m V \in \mathcal{S}$. Then $U \leqslant_m V$ is derivable. We reason by cases on the last rule applied to derive this judgment.

Case [FSM-END]. Then $U = V = \pi\mathsf{end}$ and there is nothing left to prove since [FSM-END] has no premises.

Case [FSM-TAG-IN]. Then $U = \sum_{i \in I} \mathsf{p}?\mathsf{m}_i.U_i$ and $V = \sum_{i \in J} \mathsf{p}?\mathsf{m}_i.V_i$ and $I \subseteq J$ and $U_i \leqslant_{n_i} V_i$ and $n_i \leq m$ for every $i \in I$. By definition of $\mathcal{S}$ we have that, for every $i \in I$, there exists $m_i \leq n_i$ such that $U \leqslant_{m_i} V_i \in \mathcal{S}$. Then $U \leqslant_m V$ is derivable by [FSM-TAG-IN] using premises in $\mathcal{S}$.

Case [FSM-TAG-OUT-1]. Analogous to the previous case.

Case [FSM-TAG-OUT-2]. Then $U = \sum_{i \in I} \mathsf{p}!\mathsf{m}_i.U_i$ and $V = \sum_{i \in J} \mathsf{p}!\mathsf{m}_i.V_i$ and $J \subseteq I$ and $U_i \leqslant_{n_i} V_i$ for every $i \in J$ and $n_k < m$ for some $k \in I$. By definition of $\mathcal{S}$ we have that, for every $i \in J$, there exists $m_i \leq n_i$ such that $U \leqslant_{m_i} V_i \in \mathcal{S}$. In particular, $m_k \leq n_k < m$. Then $U \leqslant_m V$ is derivable by [FSM-TAG-OUT-2] using premises in $\mathcal{S}$. $\square$

Proof of Theorem 3.2.1. The proof that $\leqslant$ is reflexive is trivial, since $S \leqslant_n S$ is derivable for every n. Concerning transitivity, by lemma 3.2.1 it suffices to show that each judgment in the set

$$\mathcal{S} \overset{\mathsf{def}}{=} \{S \leqslant_{n_1+(1+m)n_2} T \mid S \leqslant_{n_1}^m U \wedge U \leqslant_{n_2} T\}$$

is derivable using the rules in Figure 3.3 from premises that are also in $\mathcal{S}$. Suppose $S \leqslant_n T \in \mathcal{S}$. Then there exist U, n_1, m and n_2 such that $S \leqslant_{n_1}^m U$ and $U \leqslant_{n_2} T$ and $n = n_1 + (1+m)n_2$. We reason by cases on the last rules applied to derive $S \leqslant_{n_1} U$ and $U \leqslant_{n_2} T$.

Case [FSM-END]. Then $S = U = T = \pi\mathsf{end}$ hence $S \leqslant_n T$ is derivable by [FSM-END].

Case [FSM-TAG-IN]. Then $S = \sum_{i \in I} \mathsf{p}?\mathsf{m}_i.S_i$ and $U = \sum_{i \in J} \mathsf{p}?\mathsf{m}_i.U_i$ and $T = \sum_{i \in K} \mathsf{p}?\mathsf{m}_i.T_i$ and $I \subseteq J \subseteq K$ and $S_i \leqslant_{n_{1i}}^m U_i$ and $n_{1i} \leq n_1$ for every $i \in I$ and $U_i \leqslant_{n_{2i}} T_i$ and $n_{2i} \leq n_2$ for every $i \in J$. By definition of $\mathcal{S}$ we have that $S_i \leqslant_{n_{1i}+(1+m)n_{2i}} T_i \in \mathcal{S}$ for every $i \in I$. Observe that $n_{1i} + (1+m)n_{2i} \leq n_1 + (1+m)n_2 = n$ for every $i \in I$ hence $S \leqslant_n T$ is derivable by [FSM-TAG-IN].

Case [FSM-TAG-OUT-1]. Then $S = \sum_{i \in I} \mathsf{p}!\mathsf{m}_i.S_i$ and $U = \sum_{i \in I} \mathsf{p}!\mathsf{m}_i.U_i$ and $T = \sum_{i \in I} \mathsf{p}!\mathsf{m}_i.T_i$ and $S_i \leqslant_{n_{1i}}^m U_i$ and $n_{1i} \leq n_1$ and $U_i \leqslant_{n_{2i}} T_i$ and $n_{2i} \leq n_2$ for every $i \in I$. By definition of $\mathcal{S}$ we have that $S_i \leqslant_{n_{1i}+(1+m)n_{2i}} T_i \in \mathcal{S}$

for every $i \in I$. Observe that $n_{1i} + (1+m)n_{2i} \leq n_1 + (1+m)n_2 = n$ for every $i \in I$ hence $S \leqslant_n T$ is derivable by [FSM-TAG-OUT-1].

Case [FSM-TAG-OUT-1] *and* [FSM-TAG-OUT-2]. Then $S = \sum_{i\in I} \mathsf{p}!\mathsf{m}_i.S_i$ and $U = \sum_{i\in I} \mathsf{p}!\mathsf{m}_i.U_i$ and $T = \sum_{i\in J} \mathsf{p}!\mathsf{m}_i.T_i$ and $J \subseteq I$ and $S_i \leqslant^m_{n_{1i}} U_i$ and $n_{1i} \leq n_1$ for every $i \in I$ and $U_i \leqslant_{n_{2i}} T_i$ for every $i \in J$ and $n_{2k} < n_2$ for some $k \in J$. By definition of $\mathcal{S}$ we have that $S_i \leqslant_{n_{1i}+(1+m)n_{2i}} T_i \in \mathcal{S}$ for every $i \in J$. Observe that $n_{1k} + (1+m)n_{2k} < n_1 + (1+m)n_2 = n$ hence $S \leqslant_n T$ is derivable by [F-TAG-OUT-2].

Case [FSM-TAG-OUT-2] *and* [FSM-TAG-OUT-1]. Then $S = \sum_{i\in I} \mathsf{p}!\mathsf{m}_i.S_i$ and $U = \sum_{i\in J} \mathsf{p}!\mathsf{m}_i.U_i$ and $T = \sum_{i\in J} \mathsf{p}!\mathsf{m}_i.T_i$ and $J \subseteq I$ and $S_i \leqslant^m_{n_{1i}} U_i$ for every $i \in J$ and $n_{1k} < n_1$ for some $k \in J$ and $U_i \leqslant_{n_{2i}} T_i$ for every $i \in J$ and $n_{2i} \leq n_2$ for every $i \in J$. By definition of $\mathcal{S}$ we have that $S_i \leqslant_{n_{1i}+(1+m)n_{2i}} T_i \in \mathcal{S}$ for every $i \in J$. Observe that $n_{1k} + (1+m)n_{2k} < n_1 + (1+m)n_2 = n$ hence $S \leqslant_n T$ is derivable by [FSM-TAG-OUT-2].

Case [FSM-TAG-OUT-2]. Then $S = \sum_{i\in I} \mathsf{p}!\mathsf{m}_i.S_i$, $U = \sum_{i\in J} \mathsf{p}!\mathsf{m}_i.U_i$ and $T = \sum_{i\in K} \mathsf{p}!\mathsf{m}_i.T_i$ and $K \subseteq J \subseteq I$ and $S_i \leqslant^m_{n_{1i}} U_i$ for every $i \in J$ and $n_{1j} < n_1$ for some $j \in J$ and $U_i \leqslant_{n_{2i}} T_i$ for every $i \in K$ and $n_{2k} < n_2$ for some $k \in K$. By definition of $\mathcal{S}$ we have that $S_i \leqslant_{n_{1i}+(1+m)n_{2i}} T_i \in \mathcal{S}$ for every $i \in K$. Observe that

$$\begin{array}{rcll} n_{1k} + (1+m)n_{2k} & \leq & m + (1+m)n_{2k} & \text{since } n_{1k} \leq m \\ & < & 1 + m + (1+m)n_{2k} & \\ & = & (1+m)(1+n_{2k}) & \\ & \leq & (1+m)n_2 & \text{since } n_{2k} < n_2 \\ & < & n_1 + (1+m)n_2 & \text{since } n_{1j} < n_1 \end{array}$$

hence $S \leqslant_n T$ is derivable by [FSM-TAG-OUT-2]. □

3.2.1 Correctness - Soundness

𝄢 Now we prove the *soundness* of $\leqslant$ with respect to $\sqsubseteq$ (Definition 3.2.1). That is, we prove that $\leqslant$ is *coherence*-preserving just like $\sqsubseteq$ is. The proof of this result relies on a key property of $\leqslant$ not enjoyed by the usual subtyping relation on session types [Gay and Hole, 2005]: when $S \leqslant T$ and $M \mid \mathsf{p} \triangleright S$ is coherent, the session map $M \mid \mathsf{p} \triangleright T$ can successfully terminate (Lemma 3.2.3). The rank annotation on subtyping judgements is used to set up an appropriate inductive argument for proving this property.

Theorem 3.2.2 (Soundness)**.** *If $S \leqslant T$ then $S \sqsubseteq T$.*

We start with an auxiliary result formalizing the simulation entailed by the relation $S \leqslant T$.

Lemma 3.2.2. If $S \leqslant T$ and $M \mid \mathsf{p} \triangleright S$ is coherent and $M \mid \mathsf{p} \triangleright T \Rightarrow N \mid \mathsf{p} \triangleright T'$, then $M \mid \mathsf{p} \triangleright S \Rightarrow N \mid \mathsf{p} \triangleright S'$ for some $S' \leqslant T'$.

Proof. We prove the result for a single reduction $M \mid \mathsf{p} \triangleright T \xrightarrow{\tau} N \mid \mathsf{p} \triangleright T'$. The general statement then follows by a straightforward induction on the length of the reduction $M \mid \mathsf{p} \triangleright T \Rightarrow N \mid \mathsf{p} \triangleright T'$ using the fact that coherence is preserved by reductions.

Case $M \xrightarrow{\tau} N$. Then $T' = T$ and we conclude by taking $S' \overset{\mathsf{def}}{=} S$.

Case $\mathsf{p} \triangleright T \xrightarrow{\tau} \mathsf{p} \triangleright T''$. Then $\mathsf{p} \triangleright T = \mathsf{p} \triangleright \sum_{i \in I} \mathsf{q}!\mathsf{m}_i.T_i \xrightarrow{\tau} \mathsf{p} \triangleright \mathsf{q}!\mathsf{m}_k.T_k = \mathsf{p} \triangleright T'$ for some $k \in I$. From the hypothesis $S \leqslant T$ we deduce $S = \sum_{i \in J} \mathsf{q}!\mathsf{m}_i.S_i$ where $I \subseteq J$ and $S_i \leqslant T_i$ for every $i \in I$. Now we have $M \mid \mathsf{p} \triangleright S \xrightarrow{\tau} M \mid \mathsf{p} \triangleright \mathsf{q}!\mathsf{m}_k.S_k$ and also $\mathsf{q}!\mathsf{m}_k.S_k \leqslant T'$. We conclude by taking $S' \overset{\mathsf{def}}{=} \mathsf{q}!\mathsf{m}_k.S_k$.

Case $M \xrightarrow{\mathsf{q} \triangleright \mathsf{p}!\mathsf{m}} N$ *and* $\mathsf{p} \triangleright T \xrightarrow{\mathsf{p} \triangleright \mathsf{q}?\mathsf{m}} \mathsf{p} \triangleright T'$. Then $T = \sum_{i \in I} \mathsf{q}?\mathsf{m}_i.T_i$ and $\mathsf{m} = \mathsf{m}_k$ and $T' = T_k$ for some $k \in I$. From the hypothesis $S \leqslant T$ we deduce $S = \sum_{i \in J} \mathsf{q}?\mathsf{m}_i.S_i$ and $J \subseteq I$ and $S_i \leqslant T_i$ for every $i \in J$. From the hypothesis $M \mid \mathsf{p} \triangleright S$ coherent we deduce $k \in J$ or else the participant p would not be able to receive the m tag. We conclude by taking $S' \overset{\mathsf{def}}{=} S_k$.

Case $M \xrightarrow{\mathsf{q} \triangleright \mathsf{p}?\mathsf{m}} N$ *and* $\mathsf{p} \triangleright T \xrightarrow{\mathsf{p} \triangleright \mathsf{q}!\mathsf{m}} \mathsf{p} \triangleright T'$. Then $T = \sum_{i \in I} \mathsf{q}!\mathsf{m}_i.T_i$ and $\mathsf{m} = \mathsf{m}_k$ and $T' = T_k$ for some $k \in I$. From the hypothesis $S \leqslant T$ we deduce $S = \sum_{i \in J} \mathsf{q}!\mathsf{m}_i.S_i$ and $I \subseteq J$ and $S_i \leqslant T_i$ for every $i \in I$. We conclude by taking $S' \overset{\mathsf{def}}{=} S_k$.

Case $M \xrightarrow{\mathsf{q} \triangleright \mathsf{p}!U} N$ *and* $\mathsf{p} \triangleright T \xrightarrow{\mathsf{p} \triangleright \mathsf{q}?U} \mathsf{p} \triangleright T'$. Then $T = \mathsf{q}?U.T'$. From the hypothesis $S \leqslant T$ we deduce $S = \mathsf{q}?U.S'$ and $S' \leqslant T'$. We conclude by observing that $M \mid \mathsf{p} \triangleright S \xrightarrow{\tau} N \mid \mathsf{p} \triangleright S'$.

Case $M \xrightarrow{\mathsf{q} \triangleright \mathsf{p}?U} N$ *and* $\mathsf{p} \triangleright T \xrightarrow{\mathsf{p} \triangleright \mathsf{q}!U} \mathsf{p} \triangleright T'$. Then $T = \mathsf{q}!U.T'$. From the hypothesis $S \leqslant T$ we deduce $S = \mathsf{q}!U.S'$ and $S' \leqslant T'$. We conclude by observing that $M \mid \mathsf{p} \triangleright S \xrightarrow{\tau} N \mid \mathsf{p} \triangleright S'$. □

Next we show that $S \leqslant T$ preserves the termination of any session map that completes S into a coherent one.

Lemma 3.2.3. If $S \leqslant_n T$ and $M \mid \mathsf{p} \triangleright S$ is coherent, then $M \mid \mathsf{p} \triangleright T \overset{?\checkmark}{\Longrightarrow}$.

Proof. By induction on the lexicographically ordered tuple $(n, |\varphi|)$ where φ is any string of actions such that $M \overset{\overline{\varphi\pi\checkmark}}{\Longrightarrow}$ and $\mathsf{p} \triangleright S \overset{\varphi\pi\checkmark}{\Longrightarrow}$. We know that at least one such φ least does exist from the hypothesis $M \mid \mathsf{p} \triangleright S$ is coherent. We now reason by cases on the shape of φ.

Case $\varphi = \varepsilon$. Then $S = \pi\mathsf{end}$. From the hypothesis $S \leqslant_n T$ and [FSM-END] we deduce $T = \pi\mathsf{end}$ and we conclude $M \mid \mathsf{p} \triangleright T \overset{?\checkmark}{\Longrightarrow}$.

Case $\varphi = \mathsf{p}\triangleright\mathsf{q}?\mathsf{m}\psi$. Then $S = \sum_{i\in I}\mathsf{q}?\mathsf{m}_i.S_i$ and $\mathsf{m} = \mathsf{m}_k$ for some $k \in I$. From the hypothesis $S \leqslant_n T$ and [FSM-TAG-IN] we deduce $T = \sum_{i\in J}\mathsf{q}?\mathsf{m}_i.T_i$ and $I \subseteq J$ and $S_i \leqslant_{n_i} T_i$ and $n_i \leq n$ for every $i \in I$. We conclude using the induction hypothesis.

Case $\varphi = \mathsf{p}\triangleright\mathsf{q}!\mathsf{m}\psi$. Then $S = \sum_{i\in I}\mathsf{q}!\mathsf{m}_i.S_i$ and $\mathsf{m} = \mathsf{m}_k$ for some $k \in I$. We distinguish two sub-cases, according to the last rule used in the derivation of $S \leqslant_n T$. If the last rule was [FSM-TAG-OUT-1], then $T = \sum_{i\in I}\mathsf{q}!\mathsf{m}_i.T_i$ and $S_i \leqslant_{n_i} T_i$ and $n_i \leq n$ for every $i \in I$. In particular, $S_k \leqslant_{n_k} T_k$ and $n_k \leq n$ and we conclude using the induction hypothesis. If the last rule was [FSM-TAG-OUT-2], then $T = \sum_{i\in J}\mathsf{q}!\mathsf{m}_i.T_i$ with $J \subseteq I$ and $S_i \leqslant_{n_i} T_i$ for every $i \in J$ and $n_j < n$ for some $j \in J$. In particular, we have $S_j \leqslant_{n_j} T_j$ and we conclude using the induction hypothesis. □

Proof of Theorem 3.2.2. Consider a run $M \mid \mathsf{p} \triangleright T \Rightarrow N \mid \mathsf{p} \triangleright T'$. From Lemma 3.2.2 we deduce that there exists $S' \leqslant T'$ such that $M \mid \mathsf{p} \triangleright S \Rightarrow N \mid \mathsf{p} \triangleright S'$. From the hypothesis that $M \mid \mathsf{p} \triangleright S$ is coherent we deduce $N \mid \mathsf{p} \triangleright S'$ is also coherent. From Lemma 3.2.3 we conclude $N \mid \mathsf{p} \triangleright T' \overset{?\checkmark}{\Longrightarrow}$. □

3.2.2 Correctness - Completeness

Theorem 3.2.2 alone suffices to justify the adoption of $\leqslant$ as fair subtyping relation, but we are interested in understanding to which extent $\leqslant$ covers $\sqsubseteq$. In this respect, it is quite easy to see that there exist session types that are related by $\sqsubseteq$ but not by $\leqslant$. For example, consider $S = \mathsf{p}!\mathsf{a}.S$ and $T = \mathsf{p}?\mathsf{b}.T$ and observe that these two session types describe completely different protocols (the output of infinitely many a's in the case of S and the input of infinitely many b's in the case of T). In particular, we have $S \not\leqslant T$ and $T \not\leqslant S$ but also $S \sqsubseteq T$ and $T \sqsubseteq S$. That is, S and T are *unrelated* according to $\leqslant$ but they are *equivalent* according to $\sqsubseteq$. This equivalence is justified by the fact that there exists no coherent session map in which S and T could play any role, because none of them can ever terminate.

This discussion hints at the possibility that, if we restrict the attention to those session types that *can* terminate, which are the interesting ones as far as this work is concerned, then we can establish a tighter correspondence between $\leqslant$ and $\sqsubseteq$. We call such session types *bounded*, because they describe protocols for which termination is always within reach.

Definition 3.2.2 (Bounded session type)**.** We say that a session type is *bounded* if all of its subtrees contain a $\pi\mathsf{end}$ leaf.

Note that a *finite* session type is always bounded but not every bounded session type is finite. If we consider the reduction system in which states are session types and we have $S \to T$ if T is an immediate subtree of S, then S is bounded if and only if S is fairly terminating. Now, for the family of bounded session types we can prove a *relative completeness* result for $\leqslant$ with respect to $\sqsubseteq$.

Theorem 3.2.3 (Relative completeness)**.** *S bounded, $S \sqsubseteq T$ imply $S \leqslant T$.*

The proof of Theorem 3.2.3 is done by contradiction. We show that, for any bounded S, if $S \leqslant T$ does not hold then we can build a session map M called *discriminator* such that $M \mid \mathsf{p} \triangleright S$ is coherent and $M \mid \mathsf{p} \triangleright T$ is not, which contraddicts the hypothesis $S \sqsubseteq T$. The boundedness of S is necessary to make sure that it is always possible to find a session map N such that $N \mid \mathsf{p} \triangleright S$ is coherent.

For this proof we need some auxiliary notions and notation. First of all, we consider *unfair subtyping* $\leqslant_*$ from Figure 3.1 and we fix the invariance of the type of the channel being sent by substituting rules [US-CHANNEL-IN] and [US-CHANNEL-OUT] with

$$\text{US-CHANNEL}\quad \frac{S \leqslant_* T}{\mathsf{p}\pi U.S \leqslant_* \mathsf{p}\pi U.T}$$

It is straightforward to see that $\leqslant \subseteq \leqslant_*$. Then, we introduce some convenient notation for building session maps. To do this, we assume the existence of an arbitrary total order $<$ on the set of roles. Now, given a finite set of roles $\{\mathsf{p}_1, \dots, \mathsf{p}_n\}$ where $\mathsf{p}_1 < \cdots < \mathsf{p}_n$, we write $\{\mathsf{p}_1, \dots, \mathsf{p}_n\}!\mathsf{m}.S$ for the session type $\mathsf{p}_1!\mathsf{m} \cdots \mathsf{p}_n!\mathsf{m}.S$. Given a finite family $\{M_i\}_{i\in I}$ of session maps all having the same domain $\{\mathsf{q}\} \subseteq D \subseteq \mathsf{Roles} \setminus \{\mathsf{p}\}$, we write $\mathsf{q} \triangleright \sum_{i\in I} \mathsf{p}\pi\mathsf{m}_i.M_i$ for the session map M having domain D and such that

$$M(\mathsf{r}) \overset{\mathsf{def}}{=} \begin{cases} \sum_{i\in I} \mathsf{p}\pi\mathsf{m}_i.D \setminus \{\mathsf{q}\}!\mathsf{m}_i.M_i(\mathsf{q}) & \text{if } \mathsf{r} = \mathsf{q} \\ \sum_{i\in I} \mathsf{q}?\mathsf{m}_i.M_i(\mathsf{r}) & \text{if } \mathsf{r} \neq \mathsf{q} \end{cases}$$

for every $\mathsf{r} \in D$. As suggested by the notation, this session map realizes a conversation in which q first interacts with p by exchanging a tag m_i and then it informs all the other participants about the tag that has been exchanged. This session map has the property

$$M \xrightarrow{\mathsf{q}:\mathsf{p}\pi\mathsf{m}_k}\!\!\Rightarrow M_i$$

for every $k \in I$.

Similarly, given a session map N with domain $D \supseteq \{\mathsf{q}\}$, we write $\mathsf{q} \triangleright \mathsf{p}\pi U.N$ for the session map M with domain D such that

$$M(\mathsf{r}) \overset{\text{def}}{=} \begin{cases} \mathsf{p}\pi U.N(\mathsf{q}) & \text{if } \mathsf{r} = \mathsf{q} \\ N(\mathsf{r}) & \text{if } \mathsf{r} \in D \setminus \{\mathsf{p}, \mathsf{q}\} \end{cases}$$

for every $\mathsf{r} \in D$. Note that M has the property

$$M \xrightarrow{\mathsf{q}:\mathsf{p}\pi U} N$$

The first key step is showing that $S \sqsubseteq T$ implies $S \leqslant_* T$ when S is a bounded session type. That is, unfair subtyping is a *necessary condition* for fair subtyping to hold.

Lemma 3.2.4. If S is bounded and $S \sqsubseteq T$ then $S \leqslant_* T$.

Proof. Using the coinduction principle (Proposition 1.2.2) it suffices to show that each judgment in the set

$$\mathcal{S} \overset{\text{def}}{=} \{S \leqslant_* T \mid S \text{ is bounded and } S \sqsubseteq T\}$$

is derivable by the rules in Figure 3.1 from premises that satisfy the same property. Let $S \leqslant_* T \in \mathcal{S}$. Then S is bounded and $S \sqsubseteq T$. We reason by cases on the shape of S.

Case $S = \pi\mathsf{end}$. Consider $M \overset{\text{def}}{=} \mathsf{q} \triangleright \overline{\pi}\mathsf{end}$ and observe that $M \mid \mathsf{p} \triangleright S$ is coherent. Then $M \mid \mathsf{p} \triangleright T$ is coherent as well, which implies $T = \pi\mathsf{end}$. We conclude by observing that $\pi\mathsf{end} \leqslant_* \pi\mathsf{end}$ is derivable with [US-END].

Case $S = \sum_{i\in I} \mathsf{q}!\mathsf{m}_i.S_i$. Let $\{M_i\}_{i\in I}$ be a family of session maps such that $M_i \mid \mathsf{p} \triangleright S_i$ is coherent for every $i \in I$. Such family is guaranteed to exist from the hypothesis that S is bounded. Without loss of generality we may assume that the M_i all have the same domain $D \supseteq \{\mathsf{q}\}$. Let $M \overset{\text{def}}{=} \mathsf{q} \triangleright \sum_{i\in I} \mathsf{p}?\mathsf{m}_i.M_i$ and observe that $M \mid \mathsf{p} \triangleright S$ is coherent by definition of M. Then $M \mid \mathsf{p} \triangleright T$ is coherent as well. We deduce that $T = \sum_{i\in J} \mathsf{q}!\mathsf{m}_i.T_i$ and $J \subseteq I$ and also that $M_i \mid \mathsf{p} \triangleright T_i$ is coherent for every $i \in J$. Hence $S_i \sqsubseteq T_i$ for every $i \in J$, namely $S_i \leqslant_* T_i \in \mathcal{S}$ for every $i \in J$ by definition of $\mathcal{S}$. We conclude by observing that $S \leqslant_* T$ is derivable by [US-TAG-OUT].

Case $S = \sum_{i\in I} \mathsf{q}?\mathsf{m}_i.S_i$. Let $\{M_i\}_{i\in I}$ be a family of session maps such that $M_i \mid \mathsf{p} \triangleright S_i$ is coherent for every $i \in I$. Such family is guaranteed to exist from the hypothesis that S is bounded. Without loss of generality we may assume that the M_i all have the same domain $D \supseteq \{\mathsf{q}\}$. Let $M \overset{\text{def}}{=} \mathsf{q} \triangleright \sum_{i\in I} \mathsf{p}!\mathsf{m}_i.M_i$ and observe that $M \mid \mathsf{p} \triangleright S$ is coherent by definition of M.

We deduce that $T = \sum_{i\in J} \mathsf{q}?\mathsf{m}_i.T_i$ and $I \subseteq J$ and also that $M_i \mid \mathsf{p} \triangleright T_i$ is coherent for every $i \in I$. Hence $S_i \sqsubseteq T_i$ for every $i \in I$, namely $S_i \leqslant_* T_i \in \mathcal{S}$ for every $i \in I$ by definition of $\mathcal{S}$. We conclude by observing that $S \leqslant_* T$ is derivable by [US-TAG-IN].

Case $S = \mathsf{q}\pi U.S'$. Let N be a session map such that $N \mid \mathsf{p} \triangleright S'$ is coherent. Such N is guaranteed to exist from the hypothesis that S is bounded. Let $M \overset{\mathsf{def}}{=} \mathsf{q} \triangleright \mathsf{p}\overline{\pi}U.N$ and observe that $M \mid \mathsf{p} \triangleright S$ is coherent by definition of M. We deduce that $T = \mathsf{q}\pi U.T'$ and also that $N \mid \mathsf{p} \triangleright T'$ is coherent. Hence $S' \sqsubseteq T'$, namely $S' \leqslant_* T' \in \mathcal{S}$ by definition of $\mathcal{S}$. We conclude by observing that $S \leqslant_* T$ is derivable by [US-CHANNEL]. □

Next we show that every bounded session type may be part of a coherent session map. This result is somewhat related to the notion of *duality* in binary session type theories [Honda, 1993, Honda et al., 1998], showing that every behavior can be completed by a matching – dual – one.

Definition 3.2.3 (Duality)**.** Let $\mathsf{targets}(\cdot)$ be the function that yields the set of roles occurring in a session type, let S be a bounded session type and D be a non-empty set of roles that includes $\mathsf{targets}(S)$ but not p. Let $\mathsf{dual}_D(\mathsf{p} \triangleright S)$ be the session map corecursively defined by the following equations:

$$\begin{aligned}
\mathsf{dual}_D(\mathsf{p} \triangleright ?\mathsf{end}) &= \{\mathsf{q} \triangleright !\mathsf{end}\}_{\mathsf{q}\in D} \\
\mathsf{dual}_D(\mathsf{p} \triangleright !\mathsf{end}) &= \min D \triangleright ?\mathsf{end} \mid \{\mathsf{q} \triangleright !\mathsf{end}\}_{\mathsf{q}\in D\setminus\{\min D\}} \\
\mathsf{dual}_D(\mathsf{p} \triangleright \textstyle\sum_{i\in I} \mathsf{q}\pi\mathsf{m}_i.S_i) &= \mathsf{q} \triangleright \textstyle\sum_{i\in I} \mathsf{p}\overline{\pi}\mathsf{m}_i.\mathsf{dual}_D(\mathsf{p} \triangleright S_i) \\
\mathsf{dual}_D(\mathsf{p} \triangleright \mathsf{q}\pi U.S) &= \mathsf{q} \triangleright \mathsf{p}\overline{\pi}U.\mathsf{dual}_D(\mathsf{p} \triangleright S)
\end{aligned}$$

Lemma 3.2.5 (Duality)**.** $\mathsf{dual}_D(\mathsf{p} \triangleright S) \mid \mathsf{p} \triangleright S$ is coherent.

Proof. Follows from the definition of $\mathsf{dual}_D(\mathsf{p} \triangleright S)$. □

Now we provide an algorithmic way of computing the "difference" between two session types related by unfair subtyping.

Definition 3.2.4 (Subtyping weight)**.** Under the hypothesis $S \leqslant_* T$, let $\mathsf{wg}(S, T) \in \mathbb{N} \cup \{\infty\}$ be the least solution of the system of equations below:

$$\begin{aligned}
\mathsf{wg}(\pi\mathsf{end}, \pi\mathsf{end}) &= 0 \\
\mathsf{wg}(\textstyle\sum_{i\in I} \mathsf{p}?\mathsf{m}_i.S_i, \sum_{i\in J} \mathsf{p}?\mathsf{m}_i.T_i) &= \max_{i\in I} \mathsf{wg}(S_i, T_i) && I \subseteq J \\
\mathsf{wg}(\textstyle\sum_{i\in I} \mathsf{p}!\mathsf{m}_i.S_i, \sum_{i\in J} \mathsf{p}!\mathsf{m}_i.T_i) &= 1 + \min_{i\in J} \mathsf{wg}(S_i, T_i) && J \subsetneq I \\
\mathsf{wg}(\textstyle\sum_{i\in I} \mathsf{p}!\mathsf{m}_i.S_i, \sum_{i\in I} \mathsf{p}!\mathsf{m}_i.T_i) &= \min\{ \\
&\qquad 1 + \min_{i\in I} \mathsf{wg}(S_i, T_i), \\
&\qquad \max_{i\in I} \mathsf{wg}(S_i, T_i)\} \\
\mathsf{wg}(\mathsf{p}\pi U.S', \mathsf{p}\pi U.T') &= \mathsf{wg}(S', T')
\end{aligned}$$

To see that $\mathsf{wg}(S, T)$ is well defined, observe that the system of equations defining $\mathsf{wg}(S, T)$ under the hypothesis $S \leqslant_* T$ contains finitely many equations, say n, by regularity of S and T. The system is representable as a monotone endofunction F on the complete lattice $(\mathbb{N} \cup \{\infty\})^n$. Thus, F has a least solution of which $\mathsf{wg}(S, T)$ is a component. We call two session types S and T divergent if they are related by unfair subtyping and have infinite rank.

Definition 3.2.5 (Divergence)**.**
We write $S \uparrow T$ if $S \leqslant_* T$ and $\mathsf{wg}(S, T) = \infty$.

Lemma 3.2.6. If $S \uparrow T$ then the derivation of $S \leqslant_* T$ contains at least one application of [U-TAG-OUT] with $J \subsetneq I$ and one of the following holds:

1. $S = \sum_{i\in I} \mathsf{p}?\mathsf{m}_i.S_i$ and $T = \sum_{i\in J} \mathsf{p}?\mathsf{m}_i.T_i$ with $I \subseteq J$ and $S_k \uparrow T_k$ for some $k \in I$, or

2. $S = \sum_{i\in I} \mathsf{p}!\mathsf{m}_i.S_i$ and $T = \sum_{i\in J} \mathsf{p}!\mathsf{m}_j.T_j$ with $J \subseteq I$ and $S_i \uparrow T_i$ for every $i \in J$, or

3. $S = \mathsf{p}\pi U.S'$ and $T = \mathsf{p}\pi U.T'$ and $S' \uparrow T'$.

Proof. If the derivation of $S \leqslant_* T$ contained no application of [U-TAG-OUT] with $J \subsetneq I$ we would have $\mathsf{wg}(S, T) = 0$. Now we reason by cases on the last rule used to derive $S \leqslant_* T$.

Case [US-END]. Then $S = T = \pi\mathsf{end}$. This case is impossible because $\mathsf{wg}(\pi\mathsf{end}, \pi\mathsf{end}) = 0$ by definition.

Case [US-TAG-IN]. Then $S = \sum_{i\in I} \mathsf{p}?\mathsf{m}_i.S_i$ and $T = \sum_{i\in J} \mathsf{p}?\mathsf{m}_i.T_i$ with $I \subseteq J$ and $S_i \leqslant_* T_i$ for every $i \in I$ and $\infty = \mathsf{wg}(S, T) = \max_{i\in I} \mathsf{wg}(S_i, T_i)$. That is, $\mathsf{wg}(S_k, T_k) = \infty$ for some $k \in I$, hence we conclude $S_k \uparrow T_k$.

Case [US-TAG-OUT]. Then $S = \sum_{i\in I} \mathsf{p}!\mathsf{m}_i.S_i$ and $T = \sum_{i\in J} \mathsf{p}!\mathsf{m}_i.T_i$ with $J \subseteq I$ and $S_i \leqslant_* T_i$ for every $i \in J$. We distinguish two subcases. If $J \subsetneq I$ then $\infty = \mathsf{wg}(S, T) = 1 + \min_{i\in J} \mathsf{wg}(S_i, T_i)$, that is $\mathsf{wg}(S_i, T_i) = \infty$ for every $i \in J$. If $J = I$ then $\infty = \mathsf{wg}(S, T) = \min\{1 + \min_{i\in I} \mathsf{wg}(S_i, T_i), \max_{i\in I} \mathsf{wg}(S_i, T_i)\} \leq 1 + \min_{i\in I} \mathsf{wg}(S_i, T_i)$ and we have $\mathsf{wg}(S_i, T_i) = \infty$ for every $i \in I$. Therefore, in both cases, we conclude $S_i \uparrow T_i$ for every $i \in J$.

Case [US-CHANNEL]. Then $S = \mathsf{p}\pi U.S'$ and $T = \mathsf{p}\pi U.T'$ and $S' \leqslant_* T'$ and $\infty = \mathsf{wg}(S, T) = \mathsf{wg}(S', T')$ hence we conclude $S' \uparrow T'$. □

Finally, the key aspect of the proof of Lemma 3.2.7 is how we build the session map M such that $M \mid \mathsf{p} \triangleright S$ is coherent while $M \mid \mathsf{p} \triangleright T$ is not.

Definition 3.2.6 (Discriminator)**.** Let $\mathsf{disc}(\mathsf{p}, S, T)$ be the session map corecursively defined by the following equations:

$$
\begin{aligned}
\mathsf{disc}(\mathsf{p}, \textstyle\sum_{i\in I} \mathsf{q}?\mathsf{m}_i.S_i, \sum_{i\in J} \mathsf{q}?\mathsf{m}_i.T_i) &= \mathsf{q} \triangleright \textstyle\sum_{i\in I, S_i \uparrow T_i} \mathsf{p}!\mathsf{m}_i.\mathsf{disc}(\mathsf{p}, S_i, T_i) \\
&\qquad\qquad \text{if } I \subseteq J \\
\mathsf{disc}(\mathsf{p}, \textstyle\sum_{i\in I} \mathsf{q}!\mathsf{m}_i.S_i, \sum_{i\in J} \mathsf{q}!\mathsf{m}_i.T_i) &= \mathsf{q} \triangleright \textstyle\sum_{i\in J} \mathsf{q}?\mathsf{m}_i.\mathsf{disc}(\mathsf{p}, S_i, T_i) \\
&\quad + \textstyle\sum_{i\in I\setminus J} \mathsf{q}?\mathsf{m}_i.\mathsf{dual}_D(\mathsf{p} \triangleright S_i) \\
&\qquad\qquad \text{if } J \subseteq I \\
\mathsf{disc}(\mathsf{p}, \mathsf{q}\pi U.S, \mathsf{q}\pi U.T) &= \mathsf{q} \triangleright \mathsf{p}\overline{\pi}U.\mathsf{disc}(\mathsf{p}, S, T)
\end{aligned}
$$

Lemma 3.2.7. If S is bounded and $S \uparrow T$ then $S \not\sqsubseteq T$.

Proof. From the hypothesis that $S \uparrow T$ and Lemma 3.2.6 we deduce that the derivation of $S \leqslant_* T$ contains at least one application of [US-TAG-OUT] with $J \subsetneq I$. Consider $\mathsf{disc}(\mathsf{p}, S, T)$ from Definition 3.2.6. Note that $\mathsf{disc}(\mathsf{p}, S, T)$ always sends a subset of the labels accepted by S, it is willing to receive any label sent by S, and it can always terminate successfully when interacting with S. Note also that it terminates successfully only after receiving a label from S that T cannot sent. Therefore, we have $\mathsf{disc}(\mathsf{p}, S, T) \mid \mathsf{p} \triangleright S$ coherent and $\mathsf{disc}(\mathsf{p}, S, T) \mid \mathsf{p} \triangleright T$ incoherent, which proves $S \not\sqsubseteq T$. □

Proof of Theorem 3.2.3. Let $S \leqslant_@ T$ if $S \leqslant_* T$ and $\mathsf{wg}(U, T) < \infty$ for every judgment $U \leqslant_* V$ in the derivation of $S \leqslant_* T$. From Lemmas 3.2.4 and 3.2.7 we have that $S \sqsubseteq T$ implies $S \leqslant_@ T$. Indeed, if there is a judgment $U \leqslant_* V$ in the derivation of $S \leqslant_* T$ such that $\mathsf{wg}(U, V) = \infty$, then it is possible to build a session M such that $M \mid \mathsf{p} \triangleright S$ is coherent and $M \mid \mathsf{p} \triangleright T$ is not by induction on the minimum depth of the judgment $U \leqslant_* V$ in the derivation using the hypothesis that S is bounded and Lemma 3.2.7.

Now, using the principle of coinduction, it suffices to show that each judgment in the set

$$\mathcal{S} \overset{\mathsf{def}}{=} \{S \leqslant_{\mathsf{wg}(S,T)} T \mid S \leqslant_@ T\}$$

is derivable using one of the rules in Figure 3.3 whose premises all belong to $\mathcal{S}$. □

3.2.3 Generalized Inference System

The purely *coinductive* characterization of fair subtyping presented in Figures 3.2 and 3.3 has been first used in Ciccone et al. [2022]. Previous works [Padovani, 2013, 2016, Ciccone and Padovani, 2021a, 2022c] relied on a different characterization based on a generalized inference system

$$\frac{}{\pi\mathsf{end} \leqslant_* \pi\mathsf{end}} \qquad \frac{S \leqslant_* T}{!U.S \leqslant_* !U.T}$$

$$\frac{\forall i \in I : S_i \leqslant_* T_i}{\sum_{i\in I} ?\mathsf{m}_i.S_i \leqslant_* \sum_{i\in I\cup J} ?\mathsf{m}_i.T_i} \qquad \frac{\forall i \in I : S_i \leqslant_* T_i}{\sum_{i\in I\cup J} !\mathsf{m}_i.S_i \leqslant_* \sum_{i\in I} !\mathsf{m}_i.T_i}$$

$$\textsc{fs-converge}\ \frac{\forall \varphi \in \mathsf{paths}(S) \setminus \mathsf{paths}(T) : \exists \psi \le \varphi, \mathsf{m} : S(\psi!\mathsf{m}) \leqslant T(\psi!\mathsf{m})}{S \leqslant T}$$

Figure 3.4: Generalized inference system for fair subtyping

(see Section 1.2) that consists of the same rules of the original subtyping relation (Figure 3.1) equipped with a corule. The generalized inference system for binary session types is presented in Figure 3.4. Note the invariance in the type of the channel being exchanged according to Remark 3.2.1. In addition to $\leqslant$, we write $\leqslant_{\mathsf{ind}}$ for the relation defined by the *inductive* interpretation of the same inference system and $\leqslant_{\mathsf{coind}}$ for the relation defined by the *coinductive* interpretation of the inference system in fig. 3.4 excluding the corule [FS-CONVERGE]. We introduce the notions of *paths* and *residual* of a session type.

Definition 3.2.7 (Paths of a session type)**.** We say that φ is a *path* of S if $S \stackrel{\varphi}{\Longrightarrow}$. We write $\mathsf{paths}(S)$ for the (prefix-closed) set of paths of S, that is $\mathsf{paths}(S) \stackrel{\mathsf{def}}{=} \{\varphi \mid S \stackrel{\varphi}{\Longrightarrow}\}$.

Definition 3.2.8 (Residual of a session type)**.** The *residual* of a session type S with respect to a path $\varphi \in \mathsf{paths}(S)$, denoted by $S(\varphi)$, is the unique session type T such that $S \stackrel{\varphi}{\Longrightarrow} T$.

The subtle difference between fair and unfair subtyping is due to the corule [FS-CONVERGE]. Since this corule is somewhat obscure, we explain it gradually starting with the following observations:

1. Recall from Section 1.2 that $S \leqslant T$ implies $S \leqslant_{\mathsf{coind}} T$ and $S \leqslant_{\mathsf{ind}} T$. Hence, $\leqslant$ is a refinement of $\leqslant_{\mathsf{coind}}$ such that, for each pair of related session types S and T, there exists a *finite-depth* derivation tree for the judgment $S \leqslant T$ using the rules and possibly the corule [FS-CONVERGE].

2. When $S \leqslant_{\mathsf{coind}} T$ holds, it is not possible to establish a general correlation between $\mathsf{paths}(S)$ and $\mathsf{paths}(T)$. Indeed, [US-TAG-IN] entails that

some paths of T may not be present in S and [US-TAG-OUT] entails that some paths of S may not be present in T.

3. The judgment $S \leqslant T$ is trivially derivable using [FS-CONVERGE] if the path inclusion relation $\mathsf{paths}(S) \subseteq \mathsf{paths}(T)$ holds. Since [US-TAG-OUT] is the only rule that allows T to have fewer paths than S, we deduce that [FS-CONVERGE] limits (but does not always forbid) applications of [US-TAG-OUT].

4. In general [FS-CONVERGE] requires that, whenever a path φ of S is no longer present in T, it must be possible to find a prefix ψ of φ and an output $!\mathsf{m}$ shared by both S and T such that $S(\psi!\mathsf{m})$ and $T(\psi!\mathsf{m})$ are one step closer to the region of S and T where path inclusion holds.

The reason why path inclusion plays such an important role in the definition of fair subtyping is that a process using a channel of type T keeps using it according to T even if it is replaced by another channel of type $S \leqslant T$, without even realizing that the replacement has taken place. After all, this is what the “safe substitution principle” (Proposition 3.1.1) is based on. As a consequence, none of the paths in S that have disappeared in T will be offered to the process at the other end of the session. If there are “too few” paths in T compared to S, then the replacement might compromise the termination of the process at the other end of the session, should it crucially rely on those paths to terminate. When $S \leqslant T$ (and therefore $S \leqslant_{\mathsf{ind}} T$) holds, the corule [FS-CONVERGE] makes sure that the process using the channel of type S believing that it has type T is always at *finite distance* from the region where path inclusion between (some subtrees of) S and (the corresponding subtrees of) T holds. Moreover, this region is always reachable by means of *output actions* (those $!\mathsf{m}$ mentioned in [FS-CONVERGE]) which are performed actively by the process using the channel. In other words, the process using such channel is always able, in a finite amount of time and relying on choices and actions it can perform autonomously, to steer the interaction towards a region of the protocol where path inclusion holds, hence where a common path to session termination is guaranteed to exist.

To conclude, the generalized inference system in Figure 3.4 has the advantage of highlighting the *liveness*-preserving feature of fair subtyping by using [FS-CONVERGE]. However, the purely coinductive characterization that we gave in Section 3.2 allowed us to to provide a direct proof of Theorem 3.2.1. Indeed, in previous works [Padovani, 2013, 2016, Ciccone and Padovani, 2021a, 2022c], transitivity has been established indirectly by relating the generalized inference system of fair subtyping with its semantic definition (Definition 3.2.1).

In Part III we will characterize properties of session types mixing safety and liveness aspects. Starting from the definitions of the safety (coinductive) parts, we will show how to use corules to obtain the desired predicates. Hence, for what concerns fair subtyping, we will refer to Figure 3.4 to provide a sound and complete Agda mechanization of such predicate (see Section 8.5).

Part II

Enforcing Fair Termination *(Largo)*

Chapter 4

Fair Termination Of Binary Sessions

In this chapter we present the first type system for enforcing fair termination of *binary sessions.* Although in Chapter 5 we will present the same approach applied to multiparty ones, hence to a more general context, the binary case succeeds in highlighting the main requirements, properties and challenges of such a type system. For this reason, in this chapter we focus on paradigmatic examples in order to guide the reader across the main developed features. In Sections 5.1.3 and 5.2.2 we will show some more involved scenarios. It is worth noting that the proof technique that we adopt to prove the soundness of the type system is shared by both the binary and the multiparty case. However, the two proofs differ in some technicalities (*e.g.* deadlock freedom). The type system that we present in this chapter is a refined version of the one used by Ciccone and Padovani [2022c]. Indeed, Ciccone and Padovani [2022c] rely on the characterization of fair subtyping presented in Section 3.2.3. For the sake of uniformity with respect to Ciccone et al. [2022] on which Chapter 5 will be based on, we refer to the purely coinductive characterization presented in Section 3.2.

The chapter is organized as follows. In Section 4.1 we present the syntax and the semantics of the session based calculus on which we apply static analysis. Section 4.2 shows the type system for such calculus and introduce some additional properties that are required to enforce fair termination. Finally, in Section 4.3 we detail the soundness proof the type system.

Concerning a comparison of the type system we show in this chapter

P, Q	$::=$	done	termination	$\mid$	$A\langle\overline{x}\rangle$	invocation
	$\mid$	$\mathsf{wait}\,x.P$	signal in	$\mid$	$\mathsf{close}\,x$	signal out
	$\mid$	$x?(y).P$	channel in	$\mid$	$x!(y).P$	channel out
	$\mid$	$x\pi\{\mathsf{m}_i.P_i\}_{i\in I}$	label in/out	$\mid$	$P \oplus Q$	choice
	$\mid$	$(x)(P \mid Q)$	session	$\mid$	$\lceil x\rceil P$	cast

Figure 4.1: Syntax of processes

with respect to existing works, we delay the discussion to Chapter 5.

4.1 Calculus

In this section we introduce the calculus for binary sessions. We recall some basic notions. We use an infinite set of *channel names* ranged over by x, y, z, a set of *message tags* ranged over by m, and a set of *process names* ranged over by A, B, C. We write $\overline{x}$ for a possibly empty sequences of channels, extending this notation to other entities and we use π to range over the elements of the set $\{?, !\}$ of *polarities* to distinguish input actions (?) from output actions (!).

4.1.1 Syntax of Processes

A *program* is a finite set of definitions of the form $A(\overline{x}) \triangleq P$, at most one for each process name, where P is a *process* generated by the grammar in Figure 4.1. The process done is terminated and performs no action. The invocation $A\langle\overline{x}\rangle$ behaves as P if $A(\overline{x}) \triangleq P$ is the definition of A. When $\overline{x}$ is empty, we write A and $A \triangleq P$ instead of A and $A() \triangleq P$. The process $\mathsf{wait}\,x.P$ waits for a signal from channel x indicating that the session x is being closed and then continues as P. The process $\mathsf{close}\,x$ sends the termination signal on x. The process $x?(y).P$ receives a channel y from channel x and then continues as P. Dually, $x!(y).P$ sends y on x and then continues as P. The process $x\pi\{\mathsf{m}_i : P_i\}_{i\in I}$ exchanges a tag m_i on channel x and then continues as P_i. As for session types, we assume that the set I in these forms is always non-empty and that $i \neq j$ implies $\mathsf{m}_i \neq \mathsf{m}_j$ for every $i, j \in I$. Also, we write $x\pi\mathsf{m}_i.P_i$ instead of $x\pi\{\mathsf{m}_i : P_i\}_{i\in I}$ when I is a singleton $\{i\}$. A non-deterministic choice $P_1 \oplus P_2$ reduces to either P_1 or P_2. The annotation $k \in \{1, 2\}$ has no operational meaning, it is only used to record that P_k leads to the termination of the process and is omitted when irrelevant. A session $(x)(P \mid Q)$ is the parallel composition of P and Q connected by x. Note that composition and restriction is atomic. This approach is inspired by linear-logic based calculi; we provide more details later on. Finally, a *cast* $\lceil x \rceil P$ behaves exactly as P. This form simply records the fact that the type of x is subject to an application of fair subtyping in the typing derivation for P. As we will see in Section 4.2, we use this form to precisely account for all places in (the typing derivation of) a process where fair subtyping is used. Occasionally we write $\lceil x_1 \cdots x_n \rceil P$ for $\lceil x_1 \rceil \cdots \lceil x_n \rceil P$.

The only binders are channel inputs $x?(y).P$ and sessions $(x)(P \mid Q)$. We write $\mathsf{fn}(P)$ and $\mathsf{bn}(P)$ for the sets of free and bound channel names occurring in P and we identify processes modulo renaming of bound names. The program $\mathcal{P}$ that provides the meaning to the process names occurring in processes is often left implicit. Sometimes we write a process definition $A(\overline{x}) \triangleq P$ as a proposition or side condition, intending that such definition is part of the implicit program $\mathcal{P}$. Note that this approach differs from the one used by Scalas and Yoshida [2019] as they introduce process definition in the syntax. We adopt a different approach in order to keep the calculus as light as possible.

Example 4.1.1. *Let us revisit and complete the example we sketched in Example 2.1.1. We let the* ***buyer*** add *an* odd *number of items. We can model the whole system as the following set of process definitions:*

$$
\begin{aligned}
\mathit{Main} &\triangleq (y)((x)(\lceil x \rceil \mathit{Buyer}\langle x \rangle \mid \mathit{Seller}\langle x, y \rangle) \mid \mathit{Carrier}\langle y \rangle)\\
\mathit{Buyer}(x) &\triangleq x!\mathsf{add}.x!\{\mathsf{add} : \mathit{Buyer}\langle x \rangle, \mathsf{pay} : \mathsf{close}\, x\}\\
\mathit{Seller}(x, y) &\triangleq x?\{\mathsf{add} : \mathit{Seller}\langle x, y \rangle, \mathsf{pay} : \mathsf{wait}\, x.y!\mathsf{ship}.\mathsf{close}\, y\}\\
\mathit{Carrier}(y) &\triangleq y?\mathsf{ship}.\mathsf{wait}\, y.\mathsf{done}
\end{aligned}
$$

Note that Buyer deterministically sends add *to Seller as the first message, whereas he chooses among* add *and* pay *every other interaction. After Buyer has sent* pay*, it closes the session x with Seller. At this point, Seller sends* ship *to Carrier and closes the session y. The cast $\lceil x \rceil$ before the invocation of $\mathit{Buyer}\langle x \rangle$ in Main is meant to account for the mismatch between the behavior of the buyer, which always adds an odd number of items to the cart, and that of the business, which accepts any number of items added to the shopping cart.* ⌟

4.1.2 Operational Semantics

The operational semantics of processes is defined using a structural pre-congruence relation $\preccurlyeq$ and a reduction relation $\rightarrow$, both of which

[SB-PAR-COMM]	$(x)(P \mid Q) \preccurlyeq (x)(Q \mid P)$	
[SB-PAR-ASSOC]	$(x)(P \mid (y)(Q \mid R)) \preccurlyeq (y)((x)(P \mid Q) \mid R)$	if $x \in \mathsf{fn}(Q)$ and $y \notin \mathsf{fn}(P)$ and $x \notin \mathsf{fn}(R)$
[SB-CAST-COMM]	$\lceil x \rceil \lceil y \rceil P \preccurlyeq \lceil y \rceil \lceil x \rceil P$	
[SB-CAST-NEW]	$(x)(\lceil x \rceil P \mid Q) \preccurlyeq (x)(P \mid Q)$	
[SB-CAST-SWAP]	$(x)(\lceil y \rceil P \mid Q) \preccurlyeq \lceil y \rceil (x)(P \mid Q)$	if $x \neq y$
[SB-CALL]	$A\langle \overline{x} \rangle \preccurlyeq P$	if $A(\overline{x}) \triangleq P$

Figure 4.2: Structural pre-congruence of processes

are defined in Figures 4.2 and 4.3 and described hereafter. Rules [SB-PAR-COMM] and [SB-PAR-ASSOC] express the usual commutativity and associativity of parallel composition. In the case of [SB-PAR-ASSOC], the side condition $x \in \mathsf{fn}(Q)$ makes sure that the session $(x)(P|Q)$ we obtain on the right hand side does indeed connect P and Q through x. The other side conditions $y \notin \mathsf{fn}(P)$ and $x \notin \mathsf{fn}(R)$ will always hold when dealing with well-typed processes. Also note that [SB-PAR-ASSOC] only describes a right-to-left associativity of parallel composition and that left-to-right associativity is derivable by the chain of relations

$$\begin{aligned}(x)((y)(P \mid Q) \mid R) &\preccurlyeq (x)(R \mid (y)(P \mid Q)) \\ &\preccurlyeq (x)(R \mid (y)(Q \mid P)) \\ &\preccurlyeq (y)((x)(R \mid Q) \mid P) \\ &\preccurlyeq (y)((x)(Q \mid R) \mid P) \preccurlyeq (y)(P \mid (x)(Q \mid R))\end{aligned}$$

when $x \in \mathsf{fn}(Q)$. Axiom [SB-CAST-NEW] annihilates a cast on x nearby the binder for x, making sure that casts can only be removed and never added. Axioms [SB-CAST-COMM] and [SB-CAST-SWAP] are used to move casts closer to their binder so that they can be annihilated with [SB-CAST-NEW]. Rule [SB-CALL] unfolds process invocations to their definition.

Remark 4.1.1 (On structural pre-congruence). The reader might be wonder why we adopted a pre-congruence relation instead of a more common congruence one. The reason behind this choice is that in general it reduces the number of cases to be analyzed in some proofs (see Lemma 4.3.1) without compromising the properties of the calculus. Moreover, the adoption of a congruence relation would lead to some unrealistic cases from the computation point of view. Indeed, [SB-CAST-NEW] would introduce a reflexive application of fair subtyping from a session restriction and [SB-CALL] would fold a process call. ⌟

RB-CHOICE
$$\frac{}{P_1 \oplus P_2 \to P_k}\ k \in \{1,2\}$$

RB-SIGNAL
$$\frac{}{(x)(\mathsf{close}\, x \mid \mathsf{wait}\, x.P) \to P}$$

RB-CHANNEL
$$\frac{}{(x)(x!y.P \mid x?(y).Q) \to (x)(P \mid Q)}$$

RB-PICK
$$\frac{}{(x)(x!\{\mathsf{m}_i : P_i\}_{i\in I} \mid Q) \to (x)(x!\mathsf{m}_k.P_k \mid Q)}\ k \in I, |I| > 1$$

RB-TAG
$$\frac{}{(x)(x!\mathsf{m}_k.P \mid x?\{\mathsf{m}_i : Q_i\}_{i\in I}) \to (x)(P \mid Q_k)}\ k \in I$$

RB-PAR
$$\frac{P \to Q}{(x)(P \mid R) \to (x)(Q \mid R)}$$

RB-CAST
$$\frac{P \to Q}{\lceil x \rceil P \to \lceil x \rceil Q}$$

RB-STRUCT
$$\frac{P \preccurlyeq P' \qquad P' \to Q' \qquad Q' \preccurlyeq Q}{P \to Q}$$

Figure 4.3: Reduction of processes

The reduction rules are mostly unremarkable: [RB-CHOICE] models the non-deterministic choice between alternative behaviors; [RB-PICK] models a non-trivial choice among a set of labels to send; [RB-SIGNAL], [RB-LABEL] and [RB-CHANNEL] model synchronizations between a sender (on the left hand side of the parallel composition) and a receiver (on the right hand side of the parallel composition) with [RB-SIGNAL] removing the binder of a closed session; [RB-PAR], [RB-CAST] and [RB-STRUCT] close reductions under parallel compositions, under casts and by structural pre-congruence. In the following we write $\Rightarrow$ for the reflexive, transitive closure of $\to$ and $\Rightarrow^+$ for $\Rightarrow\to$.

Example 4.1.2. *With the definitions given in Example 4.1.1, it is easy to see that there is an infinite reduction sequence starting from Main in which* *the acquirer keeps adding items to the cart:*

$$
\begin{array}{rl}
& (y)((x)(\lceil x \rceil \mathit{Buyer}\langle x\rangle \mid \mathit{Seller}\langle x, y\rangle) \mid \mathit{Carrier}\langle y\rangle) \\
\Rightarrow & (y)((x)(x!\{\mathsf{add} : \mathit{Buyer}\langle x\rangle, \mathsf{pay} : \mathsf{close}\, x\} \mid \mathit{Seller}\langle x, y\rangle) \mid \mathit{Carrier}\langle y\rangle) \\
\rightarrow & (y)((x)(x!\mathsf{add}.\mathit{Buyer}\langle x\rangle \mid \mathit{Seller}\langle x, y\rangle) \mid \mathit{Carrier}\langle y\rangle) \\
\Rightarrow & (y)((x)(\mathit{Buyer}\langle x\rangle \mid \mathit{Seller}\langle x, y\rangle) \mid \mathit{Carrier}\langle y\rangle) \\
\rightarrow & \cdots
\end{array}
$$

Nonetheless, Main is fairly terminating. For example, we have:

$$
\begin{array}{rl}
& (y)((x)(x!\{\mathsf{add} : \mathit{Buyer}\langle x\rangle, \mathsf{pay} : \mathsf{close}\, x\} \mid \mathit{Seller}\langle x, y\rangle) \mid \mathit{Carrier}\langle y\rangle) \\
\rightarrow & (y)((x)(x!\mathsf{pay}.\mathsf{close}\, x \mid \mathit{Seller}\langle x, y\rangle) \mid \mathit{Carrier}\langle y\rangle) \\
\Rightarrow & (y)((x)(\mathsf{close}\, x \mid \mathsf{wait}\, x..y!\mathsf{ship}.\mathsf{close}\, y) \mid \mathit{Carrier}\langle y\rangle) \\
\rightarrow & (y)(y!\mathsf{ship}.\mathsf{close}\, y \mid \mathit{Carrier}\langle y\rangle) \\
\Rightarrow & (y)(\mathsf{close}\, y \mid \mathsf{wait}\, y.\mathsf{done}) \\
\rightarrow & \mathsf{done}
\end{array}
$$

Note that in general it might be necessary for the acquirer to add one more item to the cart before it can send the payment to the business and the carrier receives a ship *message.* ⌟

4.2 Type System

In this section we present the type system for binary sessions that we introduced in Section 4.1. Before looking at the typing rules we motivate, through a series of examples, the key properties enforced by the type system that, taken together, guarantee fair termination. There are two families of problems that can compromise fair termination. First of all, the process (or part thereof) may be unable to reduce further but is not done. In our model, this can happen for many reasons, for example: a process attempts at sending a tag on a session that the receiver is not willing to accept; a process attempts at sending a termination signal when the receiver expects a channel; the processes at the two ends of the same session are both waiting for a message from that session. These are all examples of *safety violations*, which are prevented by any ordinary session type system. In Section 4.2.1 we focus instead on *liveness violations*. Roughly speaking, liveness is violated when a process (or part thereof) engages an infinite computation that cannot possibly terminate.

4.2.1 Boundedness Properties

In Chapter 3 we have introduced a fair subtyping relation that is liveness preserving but, as we will see in a moment, the adoption of fair subtyping alone is not enough to rule out all potential liveness violations. The type system must also enforce three properties that we call *action boundedness*, *session boundedness* and *cast boundedness* guaranteeing that the overall effort required to terminate the process is finite. In the rest of the section we describe informally these properties and we show that violating even just one of them may compromise fair process termination.

Definition 4.2.1 (Action Boundedness)**.** We say that a process is *action bounded* if there is a finite upper bound to the number of actions it has to perform in order to terminate. An action-unbounded process cannot terminate.

Example 4.2.1. *Compare the following processes*

$$A \triangleq A \oplus \mathsf{done} \qquad\qquad B \triangleq B \oplus B$$

and observe that A may always reduce to done, whereas B can only reduce forever into itself. So A is action bounded whereas B is not. ⌟

We consider a parallel composition action bounded if so are *both* processes composed in parallel. Action boundedness is a necessary condition for (fair) process termination, hence the type system must guarantee that well-typed processes are action bounded. As we will see in Section 4.2.2, this can be easily achieved by means of *typing corules* (see Section 1.2). Besides, action boundedness carries along two welcome side effects. The first one is that degenerate process definitions such as $A \triangleq A$ are not action bounded and therefore are flagged as ill typed by the type system. This guarantees that finitely many unfoldings of recursive process invocations always suffice to expose some observable process behavior. The second is that action boundedness allows us to detect recursive processes that claim to use a channel in a certain way when in fact they never do so.

Example 4.2.2. *Consider the following processes*

$$A(x, y) \triangleq x!\mathsf{a}.A\langle x, y\rangle \oplus x!\mathsf{b}.\mathsf{close}\, x \qquad\qquad B(x, y) \triangleq x!\mathsf{a}.B\langle x, y\rangle$$

where $A\langle x, y\rangle$ is action bounded and $B\langle x, y\rangle$ is not. An ordinary session type system with coinductively interpreted typing rules would accept $B\langle x, y\rangle$

regardless of y's type on the grounds that y occurs once in the body of B, hence it is "used" linearly. This is unfortunate, since y is not used in any meaningful way other than being passed as an argument of B. In A, the same linearity check promptly detects that y is not used along the path to $\mathsf{close}\, x$ *that proves the boundedness of $A\langle x, y\rangle$.* ⌟

Definition 4.2.2 (Session Boundedness)**.** We say that a process is *session bounded* if there is a finite upper bound to the number of sessions it has to create in order to terminate.

It is easy to construct non-terminating processes by chaining together an infinite number of finite (or fairly terminating) sessions.

Example 4.2.3. *Compare the following processes*

$$\begin{array}{rl} A \stackrel{\triangle}{=} & (x)(\mathsf{close}\, x \mid \mathsf{wait}\, x.A) \oplus \mathsf{done} \\ B_1 \stackrel{\triangle}{=} & (x)(\mathsf{close}\, x \mid \mathsf{wait}\, x.B_1) \end{array}$$

where A always has a possibility to terminate without creating new sessions (it is session bounded) while B_1 does not (it is session unbounded). It could be argued that B_1 is already ruled out because it is not action bounded. Indeed, while the left-hand side of the parallel composition in B_1 is finite, the right hand side is not (recall that we require both *sides of a parallel composition to admit a finite path to either* done *or* $\mathsf{close}\, x$*).* ⌟

Example 4.2.4. *Below is a slightly more complex variation of B_1 that is action bounded and session unbounded. The trick is to have a finite branch on one side of the parallel composition matched by an infinite one on the other side:*

$$B_2 \stackrel{\triangle}{=} (x)(x!\{\mathsf{a} : \mathsf{close}\, x, \mathsf{b} : \mathsf{wait}\, x.B_2\} \mid x?\{\mathsf{a} : \mathsf{wait}\, x.B_2, \mathsf{b} : \mathsf{close}\, x\})$$

⌟

Example 4.2.3 shows that a session bounded process like A may still create an unbounded number of sessions. Below is another example of session bounded process that creates unboundedly many *nested* sessions, such that the first session being created is also the last one being completed.

Example 4.2.5.

$$(x)(C\langle x\rangle \mid \mathsf{wait}\, x.\mathsf{done}) \quad C(x) \stackrel{\triangle}{=} (y)(C\langle y\rangle \mid \mathsf{wait}\, y.\mathsf{close}\, x) \oplus \mathsf{close}\, x$$

While both A and C may *create an arbitrary number of sessions, they do not* have to *do so in order to terminate. This is what sets them apart from B_1 and B_2.* ⌟

To ensure session boundedness, we check that no new sessions are found in loops that occur along inevitable paths leading to the termination of a process. For example, the creation of a new session x is inevitable in both B_1 and B_2 but it is not in A and C.

Definition 4.2.3 (Cast Boundedness)**.** We say that a process is *cast bounded* if there is a finite upper bound to the number of casts it has to perform in order to terminate.

Performing a cast means applying [SM-CAST-NEW], which corresponds to a usage of fair subtyping. The reason why cast boundedness is fundamental is that the liveness-preserving property of fair subtyping holds as long as fair subtyping is used finitely many times. Conversely, infinitely many usages of fair subtyping may have the overall effect of a single usage of unfair subtyping (see Section 3.1.1). By "infinitely many usages" we mean usages of fair subtyping that occur within a loop in a recursive process.

Remark 4.2.1 (0-weight casts)*.* Notably, cast boundedness refers to those situations in which a cast with a strictly positive weight is applied inside a loop. It might be the case that a 0-weighted cast is performed (*e.g.* no output contravariance). In this case we allow such application of fair subtyping as the liveness of the session is preserved. This is a substantial difference with respect to Ciccone and Padovani [2022c]. ⌟

Example 4.2.6. *Consider the following variants of **buyer** and **seller** from Example 2.1.1 where we can assume that the **seller** closes the session as soon as he receives a tag* pay*.*

$$(x)(\mathit{Buyer}\langle x\rangle \mid \mathit{Seller}\langle x\rangle) \qquad \begin{array}{r}\mathit{Buyer}(x) \triangleq \lceil x\rceil x!\mathsf{add}.\mathit{Buyer}\langle x\rangle \\ \mathit{Seller}(x) \triangleq x?\{\mathsf{add} : \mathit{Seller}\langle x\rangle, \mathsf{pay} : \dots\}\end{array}$$

and the session type $S_b = !\mathsf{add}.S_b + !\mathsf{pay}.!\mathsf{end}$. It can be argued that the channel x is used according to S_b in $\mathit{Buyer}(x)$ and according to $\overline{S_b}$ in $\mathit{Seller}(x)$. Indeed, the structure of $\mathit{Seller}(x)$ matches perfectly that of $\overline{S_b}$ and note that $x!\mathsf{add}.\mathit{Buyer}(x)$ uses x according to $!\mathsf{add}.S_b$, which is a fair supertype of S_b accounted for by the cast $\lceil x\rceil$ in Buyer. With this cast it is as if $\mathit{Buyer}(x)$ promises to make a choice between sending add *and sending* pay *at each iteration, but systematically favors* add *over* pay*. The overall effect of these unfulfilled promises is that the actual behavior of $\mathit{Buyer}(x)$ over x is better described by the session type $S_b^\infty = !\mathsf{add}.S_b^\infty$, which is* not *a fair supertype of S_b as we have seen in Example 3.1.2.* ⌟

Although $Buyer(x)$ could be rejected on the grounds that it is not action bounded, it is possible to find an action-bounded (but slightly more involved) variation of Example 4.2.6 in which the same phenomenon occurs.

Example 4.2.7.

$$\begin{array}{l} A(x) \triangleq \lceil x \rceil x!\mathsf{more}.x?\{\mathsf{more} : A\langle x\rangle, \mathsf{stop} : \mathsf{wait}\, x.\mathsf{done}\} \\ B(x) \triangleq x?\{\mathsf{more} : \lceil x \rceil x!\mathsf{more}.B\langle x\rangle, \mathsf{stop} : \mathsf{wait}\, x.\mathsf{done}\} \end{array}$$

both $A(x)$ and $B(x)$ have a chance to continue or to terminate the session by sending either more *or* stop*, except that they systematically favor* more *over* stop*. Now, if we consider the session types*

$$\begin{array}{lcl} S & = & !\mathsf{more}.(?\mathsf{more}.S + ?\mathsf{stop}.?\mathsf{end}) + !\mathsf{stop}.!\mathsf{end} \\ S_A & = & !\mathsf{more}.(?\mathsf{more}.S + ?\mathsf{stop}.?\mathsf{end}) \\ S_B & = & ?\mathsf{more}.!\mathsf{more}.\overline{S} + ?\mathsf{stop}.?\mathsf{end} \end{array}$$

it can be argued that $A(x)$ uses x according to S_A, which is a fair supertype of S, and that $B(x)$ uses x according to S_B, which is a fair supertype of $\overline{S}$. The casts account for the differences between S and S_A in $A(x)$ and between $\overline{S}$ and S_B in $B(x)$, but they occur within loops along paths that lead to process termination, hence A and B are not cast bounded. ⌟

It is worth discussing one last attempt to work around the problem, by moving the casts outward from within $A(x)$ and $B(x)$.

Example 4.2.8.

$$\begin{array}{l} (x)(\lceil x \rceil A\langle x\rangle \mid \lceil x \rceil B\langle x\rangle) \\ A(x) \triangleq x!\mathsf{more}.x?\{\mathsf{more} : A\langle x\rangle, \mathsf{stop} : \mathsf{wait}\, x.\mathsf{done}\} \\ B(x) \triangleq x?\{\mathsf{more} : x!\mathsf{more}.B\langle x\rangle, \mathsf{stop} : \mathsf{wait}\, x.\mathsf{done}\} \end{array}$$

Now $A(x)$ uses x according to $T_A = !\mathsf{more}.(?\mathsf{more}.T_A + ?\mathsf{stop}.?\mathsf{end})$ and $B(x)$ uses x according to $T_B = ?\mathsf{more}.!\mathsf{more}.T_B + ?\mathsf{stop}.?\mathsf{end}$, but while $S \leqslant_* T_A$ and $\overline{S} \leqslant_* T_B$ both hold neither $S \leqslant T_A$ nor $\overline{S} \leqslant T_B$ does. ⌟

In summary, the non-terminating process in Example 4.2.8 is action bounded, session bounded and cast bounded, but it is typeable only using *unfair* subtyping. As we will see in Section 4.2.2, we enforce cast boundedness using the same technique already introduced for session boundedness. That is, we check that casts do not occur along "inevitable" paths leading to recursive process invocations.

4.2.2 Typing Rules

𝄢 The typing rules resemble those of a traditional session type system but differ in a few key aspects. First of all, they establish a tighter-than-usual correspondence between types and processes so that any discrepancy between actual and expected types is accounted for by explicit casts. This way, we make sure that actions leading to the termination of a session *at the type level* are matched by corresponding actions *at the process level*, a key property used in the soundness proof of the type system. In addition, the typing rules enforce the boundedness properties informally described in the previous section. Action boundedness is enforced by specifying the typing rules as a generalized inference system and using two corules to make sure that every well-typed process is at finite distance from done or a close x. Concerning session and cast boundedness, we annotate typing judgments with a *rank*, that is an upper bound to the *weights* of casts that must be performed and of sessions that must be created in order to terminate the process in the judgment.

The typing rules are defined by the generalized inference system in Figure 4.4 and derive judgements of the form $\Gamma \vdash^n P$, meaning that P is well typed in the *typing context* Γ and has rank n. A typing context is a finite map from channels to session types written $x_1 : S_1, \dots, x_n : S_n$ or $\overline{x : S}$. We use Γ and Δ to range over typing contexts, we write $\emptyset$ for the empty context and Γ, Δ for the union of Γ and Δ when they have disjoint domains. We type check a program $\{A_i(\overline{x_i}) \triangleq P_i\}_{i \in I}$ under a global set of type assignments $\{A_i : [\overline{S_i}; n_i]\}_{i \in I}$ associating each process name A_i with a tuple of session types $\overline{S_i}$ and a rank n_i. The program is well typed if $\overline{x_i : S_i} \vdash^{n_i} P_i$ for every $i \in I$, establishing that the tuple $\overline{S_i}$ corresponds to the way the channels $\overline{x_i}$ are used by P_i and that n_i is a feasible rank annotation for P_i. Hereafter, we omit the rank from judgments when it is not important.

Let us look at the typing (co)rules in detail. [TB-DONE] is the usual axiom requiring that the terminated process leaves no unused channels behind. Since done performs no casts and creates no sessions, it can have any rank. Rules [TB-WAIT] and [TB-CLOSE] concern the exchange of session termination signals. There is nothing remarkable here except noting once again that the rank of close x can be arbitrary. Rules [TB-CHANNEL-IN] and [TB-CHANNEL-OUT] are similar, but they concern the exchange of channels. Note that, in [TB-CHANNEL-OUT], the type T of the message y is required to match *exactly* that in the type of the channel x used for the communication, whereas [Gay and Hole, 2005] allow the type of y to be a subtype of T. This is one instance of the "tight correspondence" that we mentioned earlier (see

$$\textsc{tb-done}\ \frac{}{\emptyset \vdash^n \mathsf{done}} \qquad \textsc{tb-wait}\ \frac{\Gamma \vdash^n P}{\Gamma, x : ?\mathsf{end} \vdash^n \mathsf{wait}\, x.P} \qquad \textsc{tb-close}\ \frac{}{x : !\mathsf{end} \vdash^n \mathsf{close}\, x}$$

$$\textsc{tb-channel-in}\ \frac{\Gamma, x : S, y : T \vdash^n P}{\Gamma, x : ?T.S \vdash^n x?(y).P} \qquad \textsc{tb-channel-out}\ \frac{\Gamma, x : S \vdash^n P}{\Gamma, x : !T.S, y : T \vdash^n x!y.P}$$

$$\textsc{tb-tag}\ \frac{\forall i \in I : \Gamma, x : S_i \vdash^n P_i}{\Gamma, x : \pi\{\mathsf{m}_i : S_i\}_{i\in I} \vdash^n x\pi\{\mathsf{m}_i : P_i\}_{i\in I}} \qquad \textsc{tb-choice}\ \frac{\Gamma \vdash^{n_1} P \qquad \Gamma \vdash^{n_2} Q}{\Gamma \vdash^{n_k} P \oplus_k Q}\ k \in \{1,2\}$$

$$\textsc{tb-cast}\ \frac{\Gamma, x : T \vdash^n P}{\Gamma, x : S \vdash^{n+m} \lceil x \rceil P}\ S \leqslant_m T \qquad \textsc{tb-par}\ \frac{\Gamma, x : S \vdash^m P \qquad \Delta, x : T \vdash^n Q}{\Gamma, \Delta \vdash^{1+m+n} (x)(P \mid Q)}\ S \sim T$$

$$\textsc{tb-call}\ \frac{\overline{x : S} \vdash^n P}{\overline{x : S} \vdash^{m+n} A\langle \overline{x} \rangle}\ A : [\overline{S}; n], A(\overline{x}) \triangleq P$$

$$\textsc{cob-tag}\ \frac{\Gamma, x : S_k \vdash^n P_k}{\Gamma, x : \pi\{\mathsf{m}_i : S_i\}_{i\in I} \vdash^n x\pi\{\mathsf{m}_i : P_i\}_{i\in I}}\ k \in I \qquad \textsc{cob-choice}\ \frac{\Gamma \vdash^n P_k}{\Gamma \vdash^n P_1 \oplus_k P_2}$$

Figure 4.4: Typing rules

Example 4.2.11). The rule [TB-LABEL] deals with the input/output of labels. As usual, any channel other than the one affected by the communication must be used in exactly the same way in every branch. However, the rule is stricter than that of Gay and Hole [2005] because it requires an exact correspondence between the labels that can be exchanged on x by the process and those in the type of x. The fact that a conclusion and premises are all annotated with the same rank n means that n is an upper bound for the rank of all branches of a label input/output. The corule [COB-LABEL] does not impose additional constraints compared to [TB-LABEL] and has *exactly one premise*, corresponding to one branch of the process in the conclusion. The effect of [COB-LABEL], when interpreted inductively together with the other rules, is to ensure the existence of a finite typing derivation whose leaves are applications of [TB-DONE] or [TB-CLOSE], hence action boundedness.

Rule [TB-CHOICE] is a standard typing rule for non-deterministic choices, requiring that both branches are well typed in exactly the same typing context. Notice that the rank of a choice $P_1 \oplus_k P_2$ is determined by the branch indexed by the k annotation, which is elected as the branch that leads to termination. Like [COB-LABEL], the associated corule [COB-CHOICE] ensures that the same branch gets closer to done or a close x to enforce action boundedness. Without this corule, it would not be possible to find a *finite-depth* derivation tree for an action-bounded process such as A in Example 4.2.1. Coherently with [TB-CHOICE], the same branch that leads to termination is also the one that determines the rank of the choice as a whole.

Rule [TB-CAST] is Liskov's substitution principle formulated as an inference rule. It states that a channel x of type S can be safely used where a channel of type T is expected, provided that $S \leqslant T$. The most important detail to notice here is that the rank of a cast is the *weight* of the subtyping judgment plus that of the process in which the cast has effect. This way we account for this cast in the rank of the process so as to guarantee cast boundedness. Rule [TB-PAR] concerns parallel composition and session creation. The rule is shaped after the cut rule of linear logic also adopted in other session type systems based on linear logic [Caires et al., 2016, Wadler, 2014, Lindley and Morris, 2016]. In particular, the parallel processes P and Q share no channel other than the session x that connects them, so as to prevent mutual dependencies between sessions and guarantee deadlock freedom. The side condition $S \sim T$ requires that the way in which P and Q use channel x is such that the session x can fairly terminate (see Definition 2.3.1). We *do not* require that S and T are dual to each other because reductions (see [RB-PICK]) and structural pre-congruence (see [SB-CAST-NEW]) do not necessarily preserve session type duality. Also, duality does not always imply compatibility. The rank of a parallel composition is one plus that of the composed processes. By accounting for each occurrence of parallel compositions in the rank, we guarantee that well-typed processes are session bounded.

Finally, rule [TB-CALL] states that a process invocation $A\langle\overline{x}\rangle$ is well typed provided that the types associated with $\overline{x}$ match those of the global assignment $A : [\overline{S}; n]$. Note that [TB-CALL] is *not* an axiom: its premise (re)checks that the body P in the definition of A is coherent with the global type assignment $A : [\overline{S}; n]$. With this formulation of [TB-CALL], the only axioms are [TB-DONE] and [TB-CLOSE] so that the inductive interpretation of the typing (co)rules ensures action boundedness. Note also that the rank of the conclusion may be greater than the rank n associated with A. This overapproximation grants more flexibility when typing different branches in

[TB-LABEL].

Remark 4.2.2 (On structural pre-congruence...continuation). Now we have all the ingredients to understand why the choice of a pre-congruence over a congruence relation is just a design one (see Remark 4.1.1). Indeed, the such choice was compulsory in Ciccone and Padovani [2022c]. As mentioned before, in such work we relied on the characterization of fair subtyping based on a generalized inference system (see Section 3.2.3) and the subsumption rule [TB-CAST] always increased the *rank* by one. This way, [SB-CAST-NEW] interpreted in a congruence way would increase the rank of the process due to the introduced cast. Using the actual notions, a reflexive application of fair subtyping has weight zero. ⌟

Well-typed processes enjoy the expected properties, including typing preservation under structural pre-congruence and reduction. Most importantly, they fairly terminate:

Theorem 4.2.1. *Soundness If $\emptyset \vdash^n P$ and $P \Rightarrow Q$, then $Q \Rightarrow\preccurlyeq \mathsf{done}$.*

The proof of Theorem 4.2.1 follows Theorem 2.2.1. Moreover the proof that all the reducts of a process are *weakly terminating* (see Lemma 4.3.13) is loosely based on the method of helpful directions [Francez, 1986], namely on the property that a (well-typed) process *may* reduce in such a way that its measure strictly decreases (see Lemma 4.3.12). Recall that this property is not true for every reduction.

There are several valuable implications of Theorem 5.2.1 on a well-typed, closed process P:

Deadlock freedom. If Q cannot reduce any further, then it must be done (structurally precongruent to), namely there are no residual input or output actions.

Fair termination. Under the fairness assumption, Theorem 2.2.1 assures that P eventually reduces to done. This also implies that every session created by P eventually terminates.

Junk freedom. Each message produced as P executes is eventually consumed. Indeed, if Q contains a pending message, the fact that Q may reduce to done means that some process is able to consume the message and will eventually do so under the fairness assumption.

Progress. If Q contains a sub-process with pending input/output actions, the fact that Q may reduce to done means that these actions are eventually performed.

Remark 4.2.3. The rank of a non-deterministic choice $P \oplus Q$ can usually be chosen to be the minimum among those of the branches P and Q, so that the type system can handle processes like those in example 4.2.5, which *may* create new sessions or perform casts but they need not do so in order to terminate. On the contrary, the rank of a label output $x!\{\mathsf{m}_i : P_i\}_{i \in I}$ has to be an upper bound of that of all branches P_i. The motivation for such different ways of determining the rank of these process forms, despite both represent an *internal choice*, lies in the proof of Lemma 4.3.12. In $P \oplus Q$, both branches are typed in *exactly the same* typing context, meaning that the choice of one branch or the other has no substantial impact on the shortest paths that terminate the sessions used by P and Q. Thus, the "helpful" reduction can be solely driven by the rank of the chosen branch. In a label output $x!\{\mathsf{m}_i : P_i\}_{i \in I}$ it could happen that all branches with minimum rank increase the length of the shortest path that leads to the termination of x. In this case, the choice of the "helpful" reduction must prioritize the termination of x, but then the rank of the whole process has to be an upper bound of that of the branches to be sure that the measure of the reduct decreases. ⌟

4.2.3 Examples

In this section we present some examples of well-typed processes using the type system introduced in the previous section. In particular, we first complete the variant (see Example 4.1.1) of the running example (see Example 2.1.1). Then, we show some examples where some of them are related to the boundedness properties that we presented in Section 4.2.1.

Example 4.2.9. *To show that the program defined in Example 4.1.1 is well typed, consider the session types S'_b and S_s, T_s, T_c (see Example 2.3.1).*

$$\begin{array}{ll} S'_b = ?\mathsf{add}.S'_b + ?\mathsf{pay}.?\mathsf{end} & S_s = !\mathsf{add}.(!\mathsf{add}.S_s + !\mathsf{pay}.!\mathsf{end}) \\ T_s = !\mathsf{ship}.!\mathsf{end} & T_c = ?\mathsf{ship}.?\mathsf{end} \end{array}$$

and the global type assignments

$$\begin{array}{ll} \mathit{Buyer} : [S'_b; 0] & \mathit{Seller} : [S_s, T_s; 0] \\ \mathit{Carrier} : [T_c; 0] & \mathit{Main} : [(); 3] \end{array}$$

We can obtain typing derivations for Buyer, Seller and Carrier using a null *rank. In particular, we derive*

$$\dfrac{\dfrac{\dfrac{\vdots}{x : S_b' \vdash^0 \mathit{Buyer}\langle x\rangle}\,[\textsc{tb-call}] \qquad \dfrac{}{x : !\mathsf{end} \vdash^0 \mathsf{close}\, x}\,[\textsc{tb-close}]}{x : !\mathsf{add}.S_b' + !\mathsf{pay}.!\mathsf{end} \vdash^0 x!\{\mathsf{add} : \mathit{Buyer}\langle x\rangle, \mathsf{pay} : \mathsf{close}\, x\}}\,[\textsc{tb-label}]}{x : S_b' \vdash^0 x!\mathsf{add}.x!\{\mathsf{add} : \mathit{Buyer}\langle x\rangle, \mathsf{pay} : \mathsf{close}\, x\}}\,[\textsc{tb-label}]$$

for the definition of Buyer and

$$\dfrac{\dfrac{\vdots}{x : S_s, y : T_s \vdash^0 \mathit{Seller}\langle x, y\rangle} \qquad \dfrac{\dfrac{\dfrac{}{y : !\mathsf{end} \vdash^0 \mathsf{close}\, y}\,[\textsc{tb-close}]}{y : T_s \vdash^0 y!\mathsf{ship}.\mathsf{close}\, y}\,[\textsc{tb-label}]}{x : !\mathsf{end}, y : T_s \vdash^0 \mathsf{wait}\, x.\dots}\,[\textsc{tb-wait}]}{x : S_s, y : T_s \vdash^0 x?\{\mathsf{add} : \mathit{Seller}\langle x, y\rangle, \mathsf{pay} : \mathsf{wait}\, x.y!\mathsf{ship}.\mathsf{close}\, y\}}\,[\textsc{tb-label}]$$

for the definition of Seller. Note that the left branch starts with an application of [TB-CALL]*. An analogous (but finite) derivation can be easily obtained for the body of process Carrier and is omitted here for space limitations. Now we have*

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{x : S_b' \vdash^0 \mathit{Buyer}\langle x\rangle}\,[\textsc{tb-call}]}{x : S_b \vdash^1 \lceil x\rceil \mathit{Buyer}\langle x\rangle}\,[\textsc{tb-cast}] \quad \vdots}{y : T_s \vdash^2 (x)(\lceil x\rceil \mathit{Buyer}\langle x\rangle \mid \mathit{Seller}\langle x, y\rangle)}\,[\textsc{tb-par}] \qquad \dfrac{\vdots}{y : T_c \vdash^0 \mathit{Carrier}\langle y\rangle}}{\emptyset \vdash^3 (y)((x)(\lceil x\rceil \mathit{Buyer}\langle x\rangle \mid \mathit{Seller}\langle x, y\rangle) \mid \mathit{Carrier}\langle y\rangle)}$$

showing that Main too is well typed. Note that S_b from Example 2.3.1 is a fair subtype of S_b' with weight 1*; indeed* [TB-CAST] *increases the rank by such amount. In all cases, we have truncated the proof trees above the applications of* [TB-CALL]*. Of course, for each judgment occurring in these proof trees, we also have to exhibit a finite proof tree possibly using* [COB-LABEL] *proving action boundedness. This can be easily achieved for the given process definitions, observing that none of Buyer, Seller and Carrier creates new sessions and that all of their typing derivations have a finite branch.* ⌟

Example 4.2.10 (Infinite sessions/casts)**.** *In this example we demonstrate that well-typed processes may still create an unbounded number of (nested) sessions. To this aim, let us consider again the process C defined in Example 4.2.5. Notice that C is a choice whose left branch creates a new session* *and whose right branch does not. For this reason, we elect the right choice as the one that leads to termination, and therefore that determines the rank of the process. We derive*

$$\dfrac{\dfrac{\dfrac{\vdots}{y : \mathsf{!end} \vdash^0 C\langle y\rangle} \qquad \dfrac{\dfrac{}{x : \mathsf{!end} \vdash^0 \mathsf{close}\,x}}{x : \mathsf{!end}, y : \mathsf{?end} \vdash^0 \mathsf{wait}\,y.\dots}}{x : \mathsf{!end} \vdash^1 (y)(C\langle y\rangle \mid \mathsf{wait}\,y.\dots)} \qquad \dfrac{}{x : \mathsf{!end} \vdash^0 \mathsf{close}\,x}}{x : \mathsf{!end} \vdash^0 (y)(C\langle y\rangle \mid \mathsf{wait}\,y.\dots) \oplus_2 \mathsf{close}\,x}$$

In a similar way, there exist well-typed processes that perform an unbounded number of casts but whose rank is finite. For example, it is easy to obtain a typing derivation for the following alternative definition of the process Buyer discussed in Example 4.1.1:

$$\mathit{Buyer}(x) \triangleq x!\mathsf{add}.(\lceil x\rceil x!\mathsf{add}.\mathit{Buyer}\langle x\rangle \oplus_2 \lceil x\rceil x!\mathsf{pay}.\mathsf{close}\,x)$$

Even though this process uses fair subtyping an unbounded number of times, the right branch of the choice has rank 1*, which is all we need to conclude that the process has rank* 1 *overall.* ⌟

4.2.4 On Higher-Order Session Types

We have defined fair subtyping in such a way that higher-order session types are *invariant* with respect to the type of the channel being exchanged (see [FS-CHANNEL]). This is a limitation compared to traditional presentations of unfair subtyping [Gay and Hole, 2005, Castagna et al., 2009, Bernardi and Hennessy, 2016], where the covariant/contravariant rules shown below are adopted ([US-CHANNEL-IN], [US-CHANNEL-OUT]):

$$\dfrac{S \leqslant_* T \qquad U \leqslant_* V}{?U.S \leqslant_* ?V.T} \qquad\qquad \dfrac{S \leqslant_* T \qquad V \leqslant_* U}{!U.S \leqslant_* !V.T}$$

The problem of these rules is that a single application of fair subtyping allowing for co-/contra-variance of higher-order session types may have the same overall effect of infinitely many applications of fair subtyping on first-order session types and, as we have seen in Section 4.2.1, unbounded applications of fair subtyping may compromise fair termination. Below is an example that illustrates the problem. The example is not large *per se*, but it is a bit contrived because it has to involve two sessions (or else there would be no need for higher-order session types), it must be bounded (or else it could be ruled out by the action/session/cast boundedness requirements) and non-terminating.

Example 4.2.11.

$$\begin{aligned} & (y)((x)(A\langle x,y\rangle \mid B\langle x\rangle) \mid B\langle y\rangle) \\ A(x,y) & \triangleq x!\mathsf{more}.x!y.B\langle x\rangle \\ B(x) & \triangleq x?\{\mathsf{more} : x?(y).A\langle y,x\rangle, \mathsf{stop} : \mathsf{wait}\, x.\mathsf{done}\} \end{aligned}$$

The process models a master $A\langle x,y\rangle$ *connected with a* primary slave $B\langle x\rangle$ *and a* secondary slave $B\langle y\rangle$ *through the sessions x and y. The interaction among the three processes proceeds in rounds. At each round, the master may decide whether to continue or stop the interaction by sending either* more *or* stop *on the session x to the primary slave. If the master decides to continue the interaction (which it does deterministically), it also delegates y to the primary slave so that, at the next round, the roles of the three processes rotate: the master is downgraded to secondary slave, the primary slave is promoted to master, and the secondary slave becomes the primary one.* ⌟

Below is a graphical representation of the network topology modeled by the process in Example 4.2.11 and of its evolution:

$$\begin{tikzcd} & A\langle x,y\rangle \arrow[dl, dash] \arrow[dr, dash] & & \Rightarrow & & B\langle x\rangle \arrow[dl, dash] & \Rightarrow & B\langle x\rangle \arrow[dr, dash] & & \Rightarrow & & A\langle y,x\rangle \arrow[dl, dash] \arrow[dr, dash] & \\ B\langle x\rangle & & B\langle y\rangle & & A\langle y,x\rangle \arrow[r, dash] & B\langle y\rangle & & B\langle y\rangle \arrow[r, dash] & A\langle x,y\rangle & & B\langle y\rangle & & B\langle x\rangle \end{tikzcd}$$

It is clear that the process in Example 4.2.11 does not terminate since there is no $\mathsf{close}\, x$ to match the $\mathsf{wait}\, x.\mathsf{done}$. It is also relatively easy to infer the types of x and y from the structure of $A(x,y)$ and $B(x)$. In particular, if we call S_A and T_A the types of x and y in $A(x,y)$ and S_B the type of x in $B(x)$ we see that these types must satisfy the equations

$$\begin{aligned} S_A & = !\mathsf{more}.!T_A.S_B \\ S_B & = ?\mathsf{more}.?S_A.T_A + ?\mathsf{stop}.?\mathsf{end} \\ T_A & = !\mathsf{more}.!T_A.S_B + !\mathsf{stop}.!\mathsf{end} \end{aligned}$$

Note that $T_A \leqslant_1 S_A$ holds because T_A and S_A differ only for the topmost output. The validity of this relation is unquestionable as it relies on the definition of fair subtyping that we have given in Section 3.2, which is invariant with respect to higher-order session types.

Remark 4.2.4. In the following we assume to have two rules for fair subtyping allowing for co-/contra-variance of channel input and output, respectively.

$$\overset{\textsc{ch-in}}{\dfrac{S \leqslant_m T \qquad U \leqslant_n V}{?U.S \leqslant_k ?V.T}} \qquad\qquad \overset{\textsc{ch-out}}{\dfrac{S \leqslant_m T \qquad V \leqslant_n U}{!U.S \leqslant_k !V.T}}$$

where $k \in F(m,n)$ concerning the input and $k \in G(m,n)$ concerning the output. Note that $F, G : \mathbb{N} \times \mathbb{N} \to \mathcal{P}_*(\mathbb{N})$. However, [TB-CHANNEL-IN] and [TB-CHANNEL-OUT] are still invariant. ⌟

If fair subtyping allowed for covariance of higher-order inputs (see [US-CHANNEL-IN], [US-CHANNEL-OUT]), then $\overline{T_A} \leqslant S_B$, $\overline{S_B} \leqslant T_A$ (along with $\overline{S_B} \leqslant S_A$ by transitivity of $\leqslant$) would also hold and we would be able to establish that the process in Example 4.2.11 is well typed, provided that casts are placed appropriately.

Example 4.2.12. *We show the derivation for the judgment $\overline{T_A} \leqslant_k S_B$ for some k.*

$$\dfrac{\dfrac{\dfrac{\vdots}{T_A \leqslant_1 S_A} \quad \dfrac{\vdots}{\overline{T_A} \leqslant_k S_B}}{!S_A.\overline{T_A} \leqslant_m !T_A.S_b}\,[\textsc{ch-out}] \qquad \dfrac{\vdots}{!\mathsf{stop}.!\mathsf{end} \leqslant_0 !\mathsf{stop}.!\mathsf{end}}}{\overline{S_B} \leqslant_1 T_A}\,[\textsc{fsb-tag-out-2}]$$

$$\dfrac{\dfrac{\dfrac{\vdots}{T_A \leqslant_1 S_A} \quad \dfrac{\vdots}{\overline{S_B} \leqslant_1 T_A}}{?T_A.\overline{S_B} \leqslant_k ?S_A.T_A}\,[\textsc{ch-in}] \qquad \dfrac{\vdots}{?\mathsf{stop}.?\mathsf{end} \leqslant_0 ?\mathsf{stop}.?\mathsf{end}}}{\overline{T_A} \leqslant_k S_B}\,[\textsc{fsb-tag-in}]$$

Note that $m \in G(1,k)$ and $k \in F(1,1)$. Hence, a solution is guaranteed to exist. For this reason, in the following we omit the weight of the subtyping judgments involving [CH-IN] *and* [CH-OUT]. *It suffices to know that such judgments are derivable.* ⌟

Example 4.2.13. *We show a version of the process in Example 4.2.11 in which we have annotated restrictions with the types $S \mid T$ of the two endpoints and casts with the target type of the channel affected by subtyping.*

$$(y : T_A \mid \overline{T_A})((x : T_A \mid \overline{T_A})(\lceil x : S_A \rceil A\langle x, y\rangle \mid \lceil x : S_B \rceil B\langle x\rangle) \mid \lceil y : S_B \rceil B\langle y\rangle)$$

We provide another formulation of such process which is well-typed as well.

$$(y : \overline{S_B} \mid S_B)((x : \overline{S_B} \mid S_B)(\lceil x : S_A \rceil \lceil y : T_A \rceil A\langle x, y\rangle \mid B\langle x\rangle) \mid B\langle y\rangle)$$

⌟

Example 4.2.14. *We show that the process in Example 4.2.13 is well typed. Below is the partial proof tree showing that $A\langle x, y\rangle$ is well typed. Each judgment is implicitly annotated with the rank* 0*:*

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{x : S_B \vdash B\langle x\rangle}\,[\textsc{tb-call}]}{x : !T_A.S_B, y : T_A \vdash x!y.B\langle x\rangle}\,[\textsc{tb-channel-out}]}{x : S_A, y : T_A \vdash x!\mathsf{more}.x!y.B\langle x\rangle}\,[\textsc{tb-tag}]}{x : S_A, y : T_A \vdash A\langle x, y\rangle}\,[\textsc{tb-call}]$$

Below is the partial proof tree showing that $B\langle x\rangle$ is well typed. Each judgment is implicitly annotated with the rank 0*:*

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{x : T_A, y : S_A \vdash A\langle y, x\rangle}\,[\textsc{tb-call}]}{x : ?S_A.T_A \vdash x?(y).A\langle y, x\rangle}\,[\textsc{tb-channel-in}] \qquad \dfrac{\dfrac{}{\emptyset \vdash \mathsf{done}}}{x : ?\mathsf{end} \vdash \mathsf{wait}\, x.\mathsf{done}}}{x : S_B \vdash x?\{\mathsf{more} : x?(y).A\langle y, x\rangle, \mathsf{stop} : \mathsf{wait}\, x.\mathsf{done}\}}}{x : S_B \vdash B\langle x\rangle}$$

Finally, here is the partial proof tree showing that the process shown in Example 4.2.13 is well typed (again, we omit rank information since [CH-IN] *and* [CH-OUT] *are used):*

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{x : T_A, y : S_A \vdash A\langle x, y\rangle}}{x : S_A, y : S_A \vdash \lceil x\rceil A\langle x, y\rangle} \qquad \dfrac{\dfrac{\vdots}{x : T_B \vdash B\langle x\rangle}}{x : \overline{S_A} \vdash \lceil x\rceil B\langle x\rangle}}{y : S_A \vdash (x)(\lceil x\rceil A\langle x, y\rangle \mid \lceil y\rceil B\langle x\rangle)} \qquad \dfrac{\dfrac{\vdots}{y : T_B \vdash B\langle y\rangle}}{y : \overline{S_A} \vdash \lceil y\rceil B\langle y\rangle}}{\emptyset \vdash (y)((x)(\lceil x\rceil A\langle x, y\rangle \mid \lceil x\rceil B\langle x\rangle) \mid \lceil y\rceil B\langle y\rangle)}$$

⌟

By restricting fair subtyping of higher-order session types to invariant inputs and outputs, the only chance we have to build a typing derivation for the process in Example 4.2.11 is by casting y each time it is delegated, either before it is sent or after it is received.

Example 4.2.15. *Consider the following variant of Example 4.2.11:*

$$\begin{aligned} A(x : U_A, y : V_A) &\stackrel{\triangle}{=} x!\mathsf{more}.\lceil y : U_A \rceil x!y.B\langle x\rangle \\ B(x : U_B) &\stackrel{\triangle}{=} x?\{\mathsf{more} : x?(y : U_A).A\langle y, x\rangle, \mathsf{stop} : \mathsf{wait}\, x.\mathsf{done}\} \end{aligned}$$

where

$$\begin{aligned} U_A &= !\mathsf{more}.!U_A.U_B \\ U_B &= ?\mathsf{more}.?U_A.V_A + ?\mathsf{stop}.?\mathsf{end} \\ V_A &= !\mathsf{more}.!U_A.U_B + !\mathsf{stop}.!\mathsf{end} \end{aligned}$$

Note that $\lceil y : U_A \rceil$ is a "first-order" cast, in the sense that the relation $V_A \leqslant_1 U_A$ holds for fair subtyping as defined in Section 3.2 without using [US-CHANNEL-IN] *or* [US-CHANNEL-OUT]*, but the cast is now placed in a region within the definition of A that prevents finding a finite rank for A.* ⌟

4.3 Proofs

In this section we detail the proof of Theorem 4.2.1. The most intriguing aspect in such proof is that a closed, well-typed process admits a reduction sequence to done. The proof technique is related to the *method of helpful directions* [Francez, 1986]: we define a well-founded *measure* for (well-typed) processes and we prove that this measure decreases strictly as the result of "helpful" reductions. In our case, the measure of a (well-typed) process P is a lexicographically ordered pair (m, n) of natural numbers such that m is an upper bound to the number of sessions that P may need to create and of weights of casts that P may need to perform *in the future* in order to terminate, whereas n is the cumulative efforts to terminate the sessions that P has created *in the past* and that are not terminated yet. We account for this effort by measuring the shortest reduction that terminates a compatible session (Definition 2.3.1). Notably, a session terminates by [RB-CLOSE]; a cast is performed when it is absorbed by the corresponding restriction, namely by [SB-CAST-NEW].

We first show two standard results, that is, typing is preserved by structural precongruence and reduction. Then, we formally introduce the measure and we characterize some *normal forms* that we need in order to achieve the proof that such measure reduces by following the right reductions.

4.3.1 Subject Reduction

We first prove that typing is preserved by structural pre-congruence. This lemma is required to prove subject reduction when dealing with the reduction under structural pre-congruence ([RB-STRUCT]).

Lemma 4.3.1. If $\Gamma \vdash^n P$ and $P \preccurlyeq Q$, then $\Gamma \vdash^m Q$ for some $m \leq n$.

Proof. The proof is by induction on the derivation of $P \preccurlyeq Q$ and by cases on the last rule applied. We only discuss a few representative cases, the remaining ones are analogous.

Case [SB-PAR-COMM]. Then $P = (x)(P_1 \mid P_2) \preccurlyeq (x)(P_2 \mid P_1) = Q$. From [TB-PAR] we deduce that there exist Γ_1, Γ_2, x, S_1, S_2, n_1 and n_2 such that:

- $\Gamma = \Gamma_1, \Gamma_2$
- $\Gamma_i, x : S_i \vdash^{n_i} P_i$ for $i = 1, 2$
- $S_1 \sim S_2$
- $n = 1 + n_1 + n_2$

We conclude $\Gamma \vdash Q$ with one application of [TB-PAR] by taking $m \overset{\mathsf{def}}{=} n$.

Case [SB-PAR-ASSOC]. Then $P = (x)(P_1 \mid (y)(P_2 \mid P_3)) \preccurlyeq (y)((x)(P_1 \mid P_2) \mid P_3) = Q$ and $x \in \mathsf{fn}(P_2)$. From [TB-PAR] we deduce that there exist Γ_1, Γ_{23}, T_1, S_1, n_1 and n_{23} such that:

- $\Gamma = \Gamma_1, \Gamma_{23}$
- $\Gamma_1, x : T_1 \vdash^{n_1} P_1$
- $\Gamma_{23}, x : S_1 \vdash^{n_{23}} (y)(P_2 \mid P_3)$
- $T_1 \sim S_1$
- $n = 1 + n_1 + n_{23}$

From [TB-PAR] we deduce that there exist Γ_2, Γ_3, T_2, S_2, n_2 and n_3 such that:

- $\Gamma_{23} = \Gamma_2, \Gamma_3$
- $\Gamma_2, x : S_1, y : T_2 \vdash^{n_2} P_2$
- $\Gamma_3, y : S_2 \vdash^{n_3} P_3$
- $T_2 \sim S_2$
- $n_{23} = 1 + n_2 + n_3$

Using [TB-PAR] we derive $\Gamma_1, \Gamma_2, y : T_2 \vdash^{1+n_1+n_2} (x)(P_1|P_2)$. We conclude $\Gamma \vdash^m (y)((x)(P_1 \mid P_2) \mid P_3)$ with another application of [TB-PAR] by taking $m \overset{\text{def}}{=} n$.

Case [SB-CAST-NEW]. Then $P = (x)(\lceil x \rceil P_1 \mid P_2) \preccurlyeq (x)(P_1 \mid P_2)$. From [TB-PAR] we deduce that there exist Γ_1, Γ_2, S_1, T, n_1 and n_2 such that:

- $\Gamma = \Gamma_1, \Gamma_2$
- $\Gamma_1, x : S_1 \vdash^{n_1} \lceil x \rceil P_1$
- $\Gamma_2, x : T \vdash^{n_2} P_2$
- $S_1 \sim T$
- $n = 1 + n_1 + n_2$

From [TB-CAST] we deduce that there exist S_2, n_3, m_x such that

- $\Gamma_1, x : S_2 \vdash^{n_3} P_1$
- $S_1 \leqslant_{m_x} S_2$
- $n_1 = m_x + n_3$

From $S_1 \sim T$ and $S_1 \leqslant S_2$ and Theorem 3.2.2 we deduce $S_2 \sim T$. We conclude with one application of [TB-PAR] by taking $m \overset{\text{def}}{=} 1 + n_3 + n_2$ and observing that

$$m \overset{\text{def}}{=} 1 + n_3 + n_2 \leq 1 + m_x + n_3 + n_2 = 1 + n_1 + n_2 = n$$

Case [SB-CAST-SWAP]. Then $P = (x)(\lceil y \rceil P_1 \mid P_2) \preccurlyeq \lceil y \rceil (x)(P_1 \mid P_2)$ and $x \neq y$. From [TB-PAR] we deduce that there exist Γ_1, S_1, S_2, T_1, n_1 and n_2 such that:

- $\Gamma = \Gamma_1, \Gamma_2, y : T_1$
- $\Gamma_1, x : S_1, y : T_1 \vdash^{n_1} \lceil y \rceil P_1$
- $\Gamma_2, x : S_2 \vdash^{n_2} P_2$
- $S_1 \sim S_2$
- $n = 1 + n_1 + n_2$

From [TB-CAST] we deduce that there exist T_2, n_3 and m_y such that

- $T_1 \leqslant_{m_y} T_2$
- $\Gamma_1, x : S_1, y : T_2 \vdash^{n_3} P_1$
- $n_1 = m_y + n_3$

We derive $\Gamma_1, \Gamma_2, y : T_2 \vdash^{1+n_3+n_2} (x)(P_1 \,|\, P_2)$ with one application of [TB-PAR] and we conclude with one application of [TB-CAST] by taking $m \overset{\mathsf{def}}{=} n$.

Case [SB-CAST-COMM]. Then $P = \lceil x \rceil \lceil y \rceil P' \preccurlyeq \lceil y \rceil \lceil x \rceil P' = Q$. We can assume $x \neq y$ or else $P = Q$ and there is nothing to prove. From [TB-CAST] we deduce that there exist Γ_1, S_1, S_2, n_1 and m_x such that

- $\Gamma = \Gamma_1, x : S_1$
- $\Gamma_1, x : S_2 \vdash^{n_1} \lceil y \rceil P'$
- $S_1 \leqslant_{m_x} S_2$
- $n = m_x + n_1$

From [TB-CAST] and the hypothesis $x \neq y$ we deduce that there exist Γ_2, T_1, T_2, n_2 and m_y such that

- $\Gamma_1 = \Gamma_2, y : T_1$
- $\Gamma_2, x : S_2, y : T_2 \vdash^{n_2} P'$
- $T_1 \leqslant_{m_y} T_2$
- $n_1 = m_y + n_2$

We derive $\Gamma_2, x : S_1, y : T_2 \vdash^{n_1} \lceil x \rceil P'$ with one application of [TB-CAST] and we conclude with another application of [TB-CAST] by taking $m \overset{\mathsf{def}}{=} n$.

Case [SB-CALL]. Then $P = A\langle \overline{x} \rangle \preccurlyeq Q$ and $A(\overline{x}) \overset{\triangle}{=} Q$. From [TB-CALL] we conclude that there exist $\overline{S}$ and m such that $A : [\overline{S}; m]$ and $\Gamma = \overline{x : S}$ and $\overline{x : S} \vdash^m Q$ and $m \leq n$. $\square$

Then we have subject reduction, stating that typing is preserved also by reductions. Note that in this case we are not able to establish a general relation between the rank of the reducible process and that of the reduct. In particular, the rank may increase.

Lemma 4.3.2 (subject reduction)**.** If $\Gamma \vdash^n P$ and $P \to Q$, then $\Gamma \vdash^m Q$ for some m.

Proof. By induction on the derivation of $P \to Q$ and by cases on the last rule applied.

Case [RB-CHOICE]. Then $P = P_1 \oplus P_2 \to P_k = Q$ where $k \in \{1, 2\}$. From [TB-CHOICE] we deduce that $\Gamma \vdash^m Q$ for some m, which is all we need to conclude.

Case [RB-PICK]. Then $P = (x)(x!\{\mathsf{m}_i : P_i\}_{i\in I}|R) \to (x)(x!\mathsf{m}_k.P_k|R) = Q$ where $k \in I$ and $|I| > 1$. From [TB-PAR] we deduce that there exist Γ_1, Γ_2, S, T, n_1 and n_2 such that

- $\Gamma = \Gamma_1, \Gamma_2$
- $\Gamma_1, x : S \vdash^{n_1} x!\{\mathsf{m}_i : P_i\}_{i\in I}$
- $\Gamma_2, x : T \vdash^{n_2} R$
- $S \sim T$
- $n = 1 + n_1 + n_2$

From [TB-TAG] we deduce that there exists a family $S_{i\in I}$ such that:

- $S = !\{\mathsf{m}_i : S_i\}_{i\in I}$
- $\Gamma_1, x : S_i \vdash^{n_1} P_i$ for every $i \in I$

From $S \sim T$ and $S \to !\mathsf{m}_k.S_k$ and Definition 2.3.1 we deduce $!\mathsf{m}_k.S_k \sim T$. We conclude with one application of [TB-TAG] and one application of [TB-PAR] by taking $m \overset{\mathsf{def}}{=} n$.

Case [RB-SIGNAL]. Then $P = (x)(\mathsf{close}\, x \mid \mathsf{wait}\, x.Q) \to Q$. From [TB-PAR], [TB-CLOSE] and [TB-WAIT] we deduce that there exist n' and m such that:

- $x : !\mathsf{end} \vdash^{n'} \mathsf{close}\, x$
- $\Gamma, x : ?\mathsf{end} \vdash^m \mathsf{wait}\, x.Q$
- $\Gamma \vdash^m Q$
- $n = 1 + n' + m$

There is nothing left to prove.

Case [RB-LABEL]. Then $P = (x)(x!\mathsf{m}_k.R|x?\{\mathsf{m}_i : Q_i\}_{i\in I}) \to (x)(R|Q_k) = Q$ with $k \in I$. From [TB-PAR] we deduce that there exist Γ_1, Γ_2, S, T, n_1 and n_2 such that:

- $\Gamma = \Gamma_1, \Gamma_2$

- $\Gamma_1, x : S \vdash^{n_1} x!\mathsf{m}_k.R$
- $\Gamma_2, x : T \vdash^{n_2} x?\{\mathsf{m}_i : Q_i\}_{i\in I}$
- $S \sim T$
- $n = 1 + n_1 + n_2$

From [TB-TAG] we deduce that there exists S_1 such that $S = !\mathsf{m}_k.S_1$ and $\Gamma_1, x : S_1 \vdash^{n_1} R$. From [TB-TAG] we deduce that there exists a family $T_{i\in I}$ such that:

- $T = ?\{\mathsf{m}_i : T_i\}_{i\in I}$
- $\Gamma_2, x : T_i \vdash^{n_2} Q_i$ for every $i \in I$

From $S \sim T$ we deduce $S_1 \sim T_k$. We conclude with one application of [TB-PAR] by taking $m \overset{\mathsf{def}}{=} n$.

Case [RB-CHANNEL]. Then $P = (x)(x!y.P' \,|\, x?(y).Q') \to (x)(P' \,|\, Q') = Q$. From [TB-PAR] we deduce that there exist Γ_1, Γ_2, S, T, n_1 and n_2 such that

- $\Gamma = \Gamma_1, \Gamma_2$
- $\Gamma_1, x : S \vdash^{n_1} x!y.P'$
- $\Gamma_2, x : T \vdash^{n_2} x?(y).Q'$
- $S \sim T$
- $n = 1 + n_1 + n_2$

From [TB-CHANNEL-OUT] we deduce that there exist Γ_1', S_1 and S_2 such that

- $\Gamma_1 = \Gamma_1', y : S_1$
- $S = !S_1.S_2$
- $\Gamma_1', x : S_2 \vdash^{n_1} P'$

From [TB-CHANNEL-IN] we deduce that there exist T_1 and T_2 such that

- $T = ?T_1.T_2$
- $\Gamma_2, x : T_2, y : T_1 \vdash^{n_2} Q'$

From $S \sim T$ we deduce $S_1 = T_1$ and $S_2 \sim T_2$. We conclude with one application of [TB-PAR] by taking $m \overset{\text{def}}{=} n$.

Case [RB-CAST]. Then $P = \lceil x \rceil P' \rightarrow \lceil x \rceil Q' = Q$ and $P' \rightarrow Q'$. From [TB-CAST] we deduce that there exist Γ', S, T, n' and m_x such that

- $\Gamma = \Gamma', x : S$
- $\Gamma', x : T \vdash^{n'} P'$
- $S \leqslant_{m_x} T$
- $n = m_x + n'$

Using the induction hypothesis on $\Gamma', x : T \vdash^{n'} P'$ and $P' \rightarrow Q'$ we derive $\Gamma', x : T \vdash^{m'} Q'$ for some m'. We conclude with one application of [TB-CAST] by taking $m \overset{\text{def}}{=} m_x + m'$.

Case [RB-PAR]. Then $P = (x)(P' \mid R) \rightarrow (x)(Q' \mid R) = Q$ and $P' \rightarrow Q'$. From [TB-PAR] we deduce that there exist Γ_1, Γ_2, S, T, n_1 and n_2 such that:

- $\Gamma = \Gamma_1, \Gamma_2$
- $\Gamma_1, x : S \vdash^{n_1} P'$
- $\Gamma_2, x : T \vdash^{n_2} R$
- $S \sim T$
- $n = 1 + n_1 + n_2$

Using the induction hypothesis we deduce that $\Gamma_1, x : S \vdash^{m_1} Q'$ for some m_1. We conclude with one application of [TB-PAR] by taking $m \overset{\text{def}}{=} m_1 + n_2$.

Case [RB-STRUCT]. Then $P \preccurlyeq P' \rightarrow Q' \preccurlyeq Q$. From Lemma 4.3.1 we deduce $\Gamma \vdash^{n'} P'$ for some $n' \leq n$. Using the induction hypothesis we deduce $\Gamma \vdash^{m'} Q'$ for some m'. We conclude using Lemma 4.3.1 once more. □

4.3.2 Measure

At the beginning on Section 4.3 we informally introduced the *measure* of a process as a lexicographically ordered pair (m, n) where the two component refer to *past* and *future*, respectively. To distinguish between past and future ofa process P we look at its structure: all sessions that occur unguarded in P have been created and are not terminated; all casts that occur in P are yet to be performed; all sessions that occur guarded in P have not been created yet. First, we need to formalize the notion of *rank* of a session which represents the minimum effort for reaching termination.

Definition 4.3.1 (rank)**.** The *rank* of S and T, written $\|S,T\|$, is the element of $\mathbb{N}\cup\{\infty\}$ defined as

$$\|S,T\| \overset{\mathsf{def}}{=} \min\{1+|\varphi| \mid \exists\varphi,\pi : S \overset{\overline{\varphi}}{\Longrightarrow} \overline{\pi}\mathsf{end}, T \overset{\varphi}{\Longrightarrow} \pi\mathsf{end}\}$$

where $|\varphi|$ denotes the length of φ, $\overline{\varphi}$ is the string of actions obtained by dualizing all the actions in φ, and we postulate that $\min\emptyset = \infty$.

Note that the rank $\|S,T\|$ is generally unrelated to the lengths of the shortest paths of S and T that lead to termination. For example, if we take $S = ?\mathsf{a}.!\mathsf{c}.?\mathsf{a}.?\mathsf{end} + ?\mathsf{b}.?\mathsf{end}$ and $T = !\mathsf{a}.(?\mathsf{c}.!\mathsf{a}.!\mathsf{end} + ?\mathsf{d}.!\mathsf{end})$ we see that the shortest path φ such that $S(\varphi) = ?\mathsf{end}$ is $?\mathsf{b}$ of length 1 and the shortest path ψ such that $T(\psi) = !\mathsf{end}$ is $!\mathsf{a}?\mathsf{d}$ of length 2, but $\|S,T\| = 4$. Indeed, both S and T must reach termination; so we have to consider the exchange of labels $\mathsf{a}, \mathsf{c}, \mathsf{a}$ and increment by one according to Definition 4.3.1.

Theorem 4.3.1. *If $U \sim S$, then*

1. $\|U,S\| \in \mathbb{N}$ *and*
2. $S \leqslant_{\mathsf{coind}} T$ *implies* $\|U,S\| \leq \|U,T\|$.

Proof. Item 1 follows from the observation that $U \sim S$ implies $U \mid S \Rightarrow \overline{\pi}\mathsf{end} \mid \pi\mathsf{end}$ for some π, hence there exists φ such that $U \overset{\overline{\varphi}}{\Longrightarrow} \overline{\pi}\mathsf{end}$ and $S \overset{\varphi}{\Longrightarrow} \pi\mathsf{end}$. Item 2 is trivial to prove if we establish that

$$\{\varphi \mid U \overset{\overline{\varphi}}{\Longrightarrow}, T \overset{\varphi}{\Longrightarrow}\} \subseteq \{\varphi \mid U \overset{\overline{\varphi}}{\Longrightarrow}, S \overset{\varphi}{\Longrightarrow}\}$$

under the hypotheses $U \sim S$ and $S \leqslant_{\mathsf{coind}} T$. We prove that $U \overset{\overline{\varphi}}{\Longrightarrow}$ and $T \overset{\varphi}{\Longrightarrow}$ implies $S \overset{\varphi}{\Longrightarrow}$ by induction on φ and by cases on its first action.

The base case $\varphi = \varepsilon$ is trivial. If $\varphi = \pi V\psi$, then $T = \pi V.T'$ for some T' and by definition of $\leqslant_{\mathsf{coind}}$ we deduce $S = \pi V.S'$ and $S' \leqslant_{\mathsf{coind}} T'$ for some S'. From $U \sim S$ we deduce $U = PolV.U'$ for some $U' \sim S'$. Using the induction hypothesis we obtain $S' \overset{\psi}{\Longrightarrow}$, which is enough to conclude $S \overset{\varphi}{\Longrightarrow}$.

If $\varphi = !\mathsf{m}\psi$, then $T = !\{l_j : T_j\}_{j\in J}$ and $l = l_k$ for some $k \in J$. By definition of $\leqslant_{\mathsf{coind}}$ we deduce $S = !\{l_i : S_i\}_{i\in I}$ for some $I \supseteq J$, hence $S \overset{!\mathsf{m}}{\Longrightarrow}$. From $U \sim S$ we deduce $U = ?\{l_k : U_k\}_{k\in K}$ for some $K \supseteq I$ and $U_k \sim S_k$. Using the induction hypothesis we obtain $S_k \overset{\psi}{\Longrightarrow}$, from which we conclude $S \overset{\varphi}{\Longrightarrow}$.

If $\varphi = ?\mathsf{m}\psi$, then $T = ?\{l_j : T_j\}_{j\in J}$ and $l = l_k$ for some $k \in J$. By definition of $\leqslant_{\mathsf{coind}}$ we deduce $S = ?\{l_i : S_i\}_{i\in I}$ for some $I \subseteq J$. From $U \sim S$

$$\text{MTB-THREAD}\quad \frac{\Gamma \vdash^{n} P}{\Gamma \vDash^{(n,0)} P} \qquad \text{MTB-CAST}\quad \frac{\Gamma, x : T \vDash^{\mu} P}{\Gamma, x : S \vDash^{\mu+(m,0)} \lceil x \rceil P}\ S \leqslant_m T$$

$$\text{MTB-PAR}\quad \frac{\Gamma, x : S \vDash^{\mu} P \qquad \Delta, x : T \vDash^{\nu} P}{\Gamma, \Delta \vDash^{\mu+\nu+(0,\|S,T\|)} (x)(P \mid Q)}\ S \sim T$$

Figure 4.5: Typing rules with measure

we deduce $U = !\{l_k : U_k\}_{k\in K}$ for some $K \subseteq I$. Also, it must be the case that $k \in K$ and $U_k \sim S_k$. Using the induction hypothesis we obtain $S_k \overset{\psi}{\Longrightarrow}$, from which we conclude $S \overset{\varphi}{\Longrightarrow}$. □

Theorem 4.3.1 shows that every usage of (fair) subtyping may increase the amount of work that is necessary to terminate a session (Item 2), although such amount is guaranteed to remain finite as long as compatibility is preserved (Item 1). This property justifies the adoption of fair subtyping over unfair subtyping, since fair subtyping preserves compatibility whereas unfair subtyping in general does not (Section 3.1.1).

To compute the measure of a process, we introduce three refined typing rules to derive judgments of the form $\Gamma \vDash^{\mu} P$, stating that P is well typed in Γ and has measure μ. Such rules are shown in Figure 4.5.

Rule [MTB-THREAD] has *lower priority* than [MTB-PAR] and [MTB-CAST], in the sense that it applies only to processes that are not a cast or a parallel composition. We call such processes *threads* and their measure is solely determined by their rank: every cast occurring in a thread is yet to be performed and every session occurring in a thread is yet to be created. Rule [MTB-PAR] states that the measure of a parallel composition is the (pointwise) sum of the measures of the composed processes, taking into account the rank of the session x by which they are connected. Finally, [MTB-CAST] states that the measure of a cast is the same measure of the process in which the cast has effect, but with the first component increased by one to account for the fact that the cast is yet to be performed.

Note that, as a well-typed process reduces, its measure may vary arbitrarily. In particular, its measure *may increase* if the process chooses to create new sessions (see the left choice of process C in Example 4.2.5) or if it picks a label that lengthens the shortest path leading to session termination (see Example 4.1.2).

Lemma 4.3.3. The following properties hold:

1. $\Gamma \vdash^n P$ implies $\Gamma \vDash^\mu P$ for some $\mu \leq (n, 0)$;

2. $\Gamma \vDash^\mu P$ implies $\Gamma \vdash^n P$ for some n such that $\mu \leq (n, 0)$.

Proof. We prove item 1 by induction on the structure of P and by cases on the last rule applied. The proof of item 2 is by induction over $\Gamma \vDash^\mu P$.

Case $P = (x)(P_1 \mid P_2)$. Then from [TB-PAR] we deduce that there exist Γ_1, Γ_2, S_1, S_2, n_1, n_2 such that:

- $\Gamma = \Gamma_1, \Gamma_2$
- $\Gamma_i, x : S_i \vdash^{n_i} P_i$ for $i = 1, 2$
- $S_1 \sim S_2$
- $n = 1 + n_1 + n_2$

Using the induction hypothesis we deduce that there exist μ_1 and μ_2 such that $\Gamma_i, x : S_i \vDash^{\mu_i} P_i$ and $\mu_i \leq (n_i, 0)$ for $i = 1, 2$. We conclude with one application of [MTB-PAR] by taking $\mu \overset{\mathsf{def}}{=} \mu_1 + \mu_2 + (0, \|S_1, S_2\|)$ and observing that $\mu < (n_1, 0) + (n_2, 0) + (1, 0) = (n, 0)$.

Case $P = \lceil x \rceil Q$. Then from [TB-CAST] we deduce that there exist Δ, S, T, n, m, m_x such that:

- $\Gamma = \Delta, x : S$
- $\Delta, x : T \vdash^m Q$
- $S \leqslant_{m_x} T$
- $n = m_x + m$

Using the induction hypothesis we deduce $\Delta, x : T \vDash^\nu Q$ for some $\nu \leq (m, 0)$. We conclude with one application of [MTB-CAST] by taking $\mu \overset{\mathsf{def}}{=} \nu + (m_x, 0)$ and observing that $\mu \leq (m, 0) + (m_x, 0) = (n, 0)$.

In all the other cases. We conclude with one application of [MTB-THREAD] by taking $\mu \overset{\mathsf{def}}{=} (n, 0)$. □

The next lemma states that structural pre-congruence does not increase the measure of a process.

Lemma 4.3.4. If $\Gamma \vDash^\mu P$ and $P \preccurlyeq Q$, then there exists $\nu \leq \mu$ such that $\Gamma \vDash^\nu Q$.

Proof. By induction on the derivation of $P \preccurlyeq Q$ and by cases on the last rule applied. We only consider the base cases.

Case [SB-PAR-COMM]. Then $P = (x)(P_1 \mid P_2) \preccurlyeq (x)(P_2 \mid P_1) = Q$. From [MTB-PAR] we deduce that there exist Γ_1, Γ_2, S, T, μ_1 and μ_2 such that

- $\Gamma = \Gamma_1, \Gamma_2$
- $\mu = \mu_1 + \mu_2 + (0, \|S, T\|)$
- $\Gamma_1, x : S \vDash^{\mu_1} P_1$
- $\Gamma_2, x : T \vDash^{\mu_2} P_2$

We conclude with one application of [MTB-PAR] by taking $\nu \overset{\mathsf{def}}{=} \mu$.

Case [SB-PAR-ASSOC]. Then $P = (x)(P_1 \,|\, (y)(Q_1 \,|\, Q_2)) \preccurlyeq (y)((x)(P_1 \,|\, Q_1) \,|\, Q_2) = Q$. From [MTB-PAR] we deduce that there exist Γ_1, Δ_1, Δ_2, μ_1, ν_1, ν_2, S_1, S_2, T_1 and T_2 such that

- $\Gamma = \Gamma_1, \Delta_1, \Delta_2$
- $\mu = \mu_1 + (\nu_1 + \nu_2 + (0, \|T_1, T_2\|)) + (0, \|S_1, S_2\|)$
- $\Gamma_1, x : S_1 \vDash^{\mu_1} P_1$
- $\Delta_1, y : T_1, x : S_2 \vDash^{\nu_1} Q_1$
- $\Delta_2, y : T_2 \vDash^{\nu_2} Q_2$

We conclude with two applications of [MTB-PAR] by taking $\nu \overset{\mathsf{def}}{=} \mu$.

Case [SB-CAST-COMM]. Then $P = \lceil x \rceil \lceil y \rceil P' \preccurlyeq \lceil x \rceil \lceil y \rceil P' = Q$. We only consider the case $x \neq y$ or else there is nothing interesting to prove. From [MTB-CAST] we deduce that there exist Γ', μ', S_1, T_1, S_2, T_2, m_x and m_y such that

- $S_1 \leqslant_{m_x} S_2$
- $T_1 \leqslant_{m_y} T_2$
- $\Gamma = \Gamma', x : S_1, y : T_1$
- $\mu = \mu' + (m_x, 0) + (m_y, 0)$
- $\Gamma', x : S_2, y : T_2 \vDash^{\mu'} P'$

We conclude with two applications of [MTB-CAST] by taking $\nu \overset{\mathsf{def}}{=} \mu$.

Case [SB-CAST-NEW]. Then $P = (x)(\lceil x \rceil P_1 \,|\, P_2) \preccurlyeq (x)(P_1 \,|\, P_2) = Q$. From [MTB-PAR] we deduce that there exist Γ_1, Γ_2, μ_1, μ_2, S_1 and T such that:

- $\Gamma = \Gamma_1, \Gamma_2$
- $\mu = \mu_1 + \mu_2 + (0, \|S_1, T\|)$
- $\Gamma_1, x : S_1 \vDash^{\mu_1} \lceil x \rceil P_1$ and $\Gamma_2, x : T \vDash^{\mu_2} P_2$
- $S_1 \sim T$

From [MTB-CAST] we deduce that there exist μ_1', S_2 and m_x such that

- $S_1 \leqslant_{m_x} S_2$
- $\mu_1 = \mu_1' + (m_x, 0)$
- $\Gamma_1, x : S_2 \vDash^{\mu_1'} P_1$

From $S_1 \sim T$ and $S_1 \leqslant_{m_x} S_2$ and Definition 2.3.1 we deduce $S_2 \sim T$. We conclude with one application of [MTB-PAR] taking $\nu \overset{\mathsf{def}}{=} \mu_1' + \mu_2 + (0, \|S_2, T\|) < \mu$.

Case [SB-CAST-SWAP]. Then $P = (x)(\lceil y \rceil P_1 \mid P_2) \preccurlyeq \lceil y \rceil (x)(P_1 \mid P_2) = Q$ and $x \neq y$. From [MTB-PAR] we deduce that there exist Γ_1, μ_1, μ_2, S_1, S_2 and T_1 such that:

- $\Gamma = \Gamma_1, \Gamma_2, y : T_1$
- $\mu = \mu_1 + \mu_2 + (0, \|S_1, S_2\|)$
- $\Gamma_1, x : S_1, y : T_1 \vDash^{\mu_1} \lceil y \rceil P_1$ and $\Gamma_2, x : S_2 \vDash^{\mu_2} P_2$
- $S_1 \sim S_2$

From [MTB-CAST] we deduce that there exist μ_1', T_2 and m_y such that

- $T_1 \leqslant_{m_y} T_2$
- $\mu_1 = \mu_1' + (m_y, 0)$
- $\Gamma_1, x : S_1, y : T_2 \vdash P_1$

We derive $\Gamma_1, \Gamma_2, y : T_2 \vDash^{\mu_1' + \mu_2 + (0, \|S_1, S_2\|)} (x)(P_1 \mid P_2)$ with one application of [MTB-PAR] and we conclude with one application of [MTB-CAST] by taking $\nu \overset{\mathsf{def}}{=} \mu$.

Case [SB-CALL]. Then $P = A\langle \overline{x} \rangle \preccurlyeq Q$ where $A(\overline{x}) \overset{\triangle}{=} Q$. From [MTB-THREAD] we deduce that $\Gamma \vdash^n A\langle \overline{x} \rangle$ for some n such that $\mu = (n, 0)$. Using Lemma 4.3.1 we deduce that $\Gamma \vdash^m Q$ for some $m \leq n$. Using Lemma 4.3.3 we deduce that $\Gamma \vDash^\nu Q$ for some $\nu \leq (m, 0)$. We conclude by observing that $\nu \leq (m, 0) \leq (n, 0) = \mu$. □

4.3.3 Normal Forms

𝄢 In this section we introduce some normal forms that are instrumental to the soundness proof of the type system. In particular, they allow us to prove a *quasi* deadlock freedom result which states that a well types process can be rearranged in such a way that one of the reduction rule is ready to be applied. Deadlock freedom is achieved by considering the process in such shape and the fact that the session under analysis is *compatible*.

To this aim, it is useful to also introduce *process contexts* as a convenient way of referring to sub-processes. A process context $\mathcal{C}$ is essentially a process with a *hole* denoted by [ ]:

$$\textbf{Process context} \quad \mathcal{C}, \mathcal{D} \quad ::= \quad [\,] \mid (x)(\mathcal{C} \mid P) \mid (x)(P \mid \mathcal{C}) \mid \lceil x \rceil \mathcal{C}$$

As usual, we write $\mathcal{C}[P]$ for the process obtained by replacing the hole in $\mathcal{C}$ with P. Note that this operation may capture channel names that occur free in P and that are bound by $\mathcal{C}$.

Definition 4.3.2 (Choice Normal Form)**.** We say that $P_1 \oplus P_2$ is an *unguarded choice* of P if there exists $\mathcal{C}$ such that $P \preccurlyeq \mathcal{C}[P_1 \oplus P_2]$. We say that P is in *choice normal form* if it has no unguarded choices.

We introduce a normal form that makes it easier to locate the components of a process that may interact with each other. Intuitively, a process is in *thread normal form* if it consists of an initial prefix of casts followed by a parallel composition of threads, where a thread is either done or a process waiting to perform an input/output action on some channel x. In this latter case, we say that the thread is an x-thread. Note that a process invocation $A\langle\overline{x}\rangle$ is *not* a thread. Formally:

Definition 4.3.3 (Thread Normal Form)**.** A process is in *thread normal form* if it is generated by the grammar below:

$$\begin{array}{rcl}
P^{nf}, Q^{nf} & ::= & \lceil x \rceil P^{nf} \mid P^{par} \\
P^{par}, Q^{par} & ::= & (x)(P^{par} \mid Q^{par}) \mid P^{th} \\
P^{th} & ::= & \mathsf{done} \mid \mathsf{close}\, x \mid \mathsf{wait}\, x.P \mid x\pi\{\mathsf{m}_i : P_i\}_{i \in I} \mid x!y.P \mid x?(y).P
\end{array}$$

At last, a process in proximity normal form is such that there exist at least two x-threads that are next to each other. Since each thread is waiting to perform an operation on the same session x, the two thread may potentially reduce if the operations are complementary ones.

Definition 4.3.4 (Proximity Normal Form)**.** We say that P^{nf} is in *proximity normal form* if $P^{nf} = \mathcal{C}[(x)(P^{th} \mid Q^{th})]$ for some $\mathcal{C}$, x, P^{th} and Q^{th} where P^{th} and Q^{th} are x-threads.

We can reduce any well-typed process into a process that is in choice normal form. The fact that the original process is well typed guarantees that this reduction eventually terminates when all the unguarded choices have been resolved.

Lemma 4.3.5. If $\Gamma \vdash^n P$ and $\Gamma \vdash_{\mathsf{ind}} P$, then there exists Q in choice normal form such that $P \Rightarrow Q$ and $\Gamma \vdash^m Q$ for some $m \leq n$.

Proof. By induction on $\Gamma \vdash_{\mathsf{ind}} P$ and by cases on the last rule applied.

Case P is already in choice normal form. We conclude taking $Q \overset{\mathsf{def}}{=} P$ and $m \overset{\mathsf{def}}{=} n$.

Case [TB-CALL]. Then $P = A\langle\overline{u}\rangle$ and $A(\overline{x}) \overset{\triangle}{=} R$. We deduce $\Gamma = \overline{u : S}$, $A : [\overline{S}; n']$ and $\Gamma \vdash_{\mathsf{ind}} R\{\overline{u}/\overline{x}\}$. Moreover, it must be the case that $\Gamma \vdash^{n'} R\{\overline{u}/\overline{x}\}$ and $n' \leq n$ since [TB-CALL] is used in the coinductive judgment as well. Using the induction hypothesis we deduce that there exist Q in choice normal form and $m \leq n'$ such that $R\{\overline{u}/\overline{x}\} \Rightarrow Q$ and $\Gamma \vdash^m Q$. We conclude by observing that $P \Rightarrow Q$ using [RB-STRUCT] and that $m \leq n' \leq n$.

Case [COB-CHOICE]. Then $P = P_1 \oplus P_2$. We deduce $\Gamma \vdash_{\mathsf{ind}} P_k$ with $k \in \{1, 2\}$. Moreover, it must be the case that $\Gamma \vdash^n P_k$ since [TB-CHOICE] is used in the coinductive judgment. Using the induction hypothesis we deduce that there exist Q in choice normal form and $m \leq n$ such that $P_k \Rightarrow Q$ and $\Gamma \vdash^m Q$. We conclude by observing that $P \to P_k$ by [RB-CHOICE].

Case [TB-CHOICE]. Analogous to the previous case but we consider the premise in which the rank is the same of the conclusion to keep sure that it does not increase.

Case [TB-PAR]. Then $P = (x)(P_1 \mid P_2)$. We deduce that there exist $\Gamma_1, \Gamma_2, S_1, S_2$ such that

- $\Gamma = \Gamma_1, \Gamma_2$
- $\Gamma_1, x : S_1 \vdash_{\mathsf{ind}} P_1$
- $\Gamma_2, x : S_2 \vdash_{\mathsf{ind}} P_2$
- $S_1 \sim S_2$

Furthermore, it must be the case that there exist n_1 and n_2 such that $\Gamma_i, x : S_i \vdash^{n_i} P_i$ for $i = 1, 2$ and $n = 1 + n_1 + n_2$ since [TB-PAR] is used in the coinductive judgment as well. Using the induction hypothesis we deduce that there exist Q_i in choice normal form and $m_i \leq n_i$ such that $P_i \Rightarrow Q_i$ and $\Gamma_i, x : S_i \vdash^{m_i} Q_i$ for $i = 1, 2$. We conclude by taking $m \overset{\mathsf{def}}{=} 1 + m_1 + m_2$ and $Q \overset{\mathsf{def}}{=} (s)(Q_1 \mid Q_2)$ with one application of [TB-PAR], observing that $m = 1 + +m_1 + m_2 \leq 1 + n_1 + n_2 = n$ and that $P \Rightarrow Q$ by [RB-PAR].

Case [TB-CAST]. Then $P = \lceil x \rceil P'$. Analogous to the previous case, just simpler. □

Lemma 4.3.6. If $\Gamma \vdash^n P$, then there exists Q in choice normal form such that $P \Rightarrow Q$ and $\Gamma \vdash^m Q$ for some $m \leq n$.

Proof.
Consequence of Lemma 4.3.5 noting that $\Gamma \vdash^n P$ implies $\Gamma \vdash_{\mathsf{ind}} P$. □

Lemma 4.3.7. If $\Gamma \vDash^\mu P$, then there exist Q in choice normal form and $\nu \leq \mu$ such that $P \Rightarrow Q$ and $\Gamma \vDash^\nu Q$.

Proof. By induction on $\Gamma \vDash^\mu P$ and by cases on the last rule applied.

Case [MTB-THREAD]. Then P is a thread. We deduce that

- $\mu = (n, 0)$ for some n
- $\Gamma \vdash^n P$

From Lemma 4.3.6 we deduce that there exist Q and $m \leq n$ such that $P \Rightarrow Q$ and $\Gamma \vdash^m Q$. From Lemma 4.3.3 we deduce $\Gamma \vDash^\nu Q$ for some $\nu \leq (m, 0)$. We conclude observing that $\nu \leq (m, 0) \leq (n, 0) = \mu$.

Case [MTB-CAST]. Then $P = \lceil x \rceil P'$. We deduce that

- $\Gamma = \Delta, x : S$
- $S \leqslant_n T$
- $\mu = \mu' + (n, 0)$
- $\Gamma', x : T \vDash^{\mu'} P'$

Using the induction hypothesis we deduce that there exist Q' and $\nu' \leq \mu'$ such that $P' \Rightarrow Q'$ and $\Gamma', x : T \vDash^{\nu'} Q'$. We conclude with an application of [MTB-CAST] taking $Q \overset{\mathsf{def}}{=} \lceil x \rceil Q'$, $\nu \overset{\mathsf{def}}{=} \nu' + (n, 0)$ and observing that $P \Rightarrow Q$ using [RB-CAST].

Case [MTB-PAR]. Then $P = (s)(P_1 \mid P_2)$. We deduce that there exist $\Gamma_1, \Gamma_2, S_1, S_2, \mu_1, \mu_2$ such that

- $\Gamma = \Gamma_1, \Gamma_2$

- $\Gamma_1, x : S_1 \vDash^{\mu_1} P_1$
- $\Gamma_2, x : S_2 \vDash^{\mu_2} P_2$
- $\mu = \mu_1 + \mu_2 + (0, \|S_1, S_2\|)$
- $S_1 \sim S_2$

Using the induction hypothesis we deduce that there exist Q_i in choice normal form and $\nu_i \leq \mu_i$ such that $P_i \Rightarrow Q_i$ and $\Gamma_i, x : S_i \vDash^{\nu_i} Q_i$ for $i = 1, 2$. We conclude by taking $\nu \overset{\mathsf{def}}{=} \nu_i + \nu_2 + (0, \|S_1, S_2\|)$ and $Q \overset{\mathsf{def}}{=} (s)(Q_1 \mid Q_2)$ with one application of [MTB-PAR], observing that

$$\nu = \nu_1 + \nu_2 + (0, \|S_1, S_2\|) \leq \mu_1 + \mu_2 + (0, \|S_1, S_2\|) = \mu$$

and that $P \Rightarrow Q$ by [RB-PAR]. □

It is easy to rewrite any *well-typed* process that is in choice normal form into thread normal form using structural pre-congruence. The hypothesis that the process is well typed, at least according to the inductive interpretation of the typing rules with the corule [COB-TAG], is necessary to guarantee that a process invocation may eventually be expanded to a term other than another process invocation. For example, the process A defined by $A \overset{\triangle}{=} A$ has no thread normal form and is ill typed. By combining this result with Lemma 4.3.1 we can deduce that the obtained thread normal form is also well typed.

Lemma 4.3.8. If $\Gamma \vdash_{\mathsf{ind}} P$ and P is in choice normal form, then there exists P^{nf} such that $P \preccurlyeq P^{nf}$.

Proof. By induction on $\Gamma \vdash_{\mathsf{ind}} P$ and by cases on the last rule applied.

Cases [TB-CHOICE] *and* [CO-CHOICE]. These cases are impossible from the hypothesis that P is in choice normal form.

Cases [TB-DONE], [TB-WAIT], [TB-CLOSE], [TB-CHANNEL-IN], [TB-CHANNEL-OUT], [TB-LABEL], [CO-LABEL]. Then P is a thread and is already in thread normal form and we conclude by reflexivity of $\preccurlyeq$.

Case [TB-CALL]. Then there exist A, Q, $\overline{x}$ and $\overline{S}$ such that

- $P = A\langle\overline{x}\rangle$
- $A(\overline{x}) \overset{\triangle}{=} Q$
- $\Gamma = \overline{x : S}$

- $\overline{x : S} \vdash_{\mathsf{ind}} Q$

Using the induction hypothesis on $\overline{x : S} \vdash_{\mathsf{ind}} Q$ we deduce that there exists P^{nf} such that $Q \preccurlyeq P^{nf}$. We conclude $P \preccurlyeq P^{nf}$ using [SB-CALL] and the transitivity of $\preccurlyeq$.

Case [TB-PAR]. Then there exist x, P_1, P_2, Γ_1, Γ_2, S_1 and S_2 such that

- $P = (x)(P_1 \mid P_2)$
- $\Gamma = \Gamma_1, \Gamma_2$
- $\Gamma_i, x : S_i \vdash_{\mathsf{ind}} P_i$ for $i = 1, 2$

Using the induction hypothesis on $\Gamma_i, x : S_i \vdash_{\mathsf{ind}} P_i$ we deduce that there exists P_i^{nf} such that $P_i \preccurlyeq P_i^{nf}$ for $i = 1, 2$. By definition of thread normal form, it must be the case that $P_i^{nf} = \lceil \overline{x_i} \rceil P_i^{par}$ for some $\overline{x_i}$ and P_i^{par}. Let $\overline{y_i}$ be the same sequence as $\overline{x_i}$ except that occurrences of x have been removed. We conclude by taking $P^{nf} \stackrel{\mathsf{def}}{=} \lceil \overline{y_1 y_2} \rceil (x)(P_1^{par} \mid P_2^{par})$ and observing that

$$\begin{array}{rcll}
P & = & (x)(P_1 \mid P_2) & \text{by definition of } P \\
 & \preccurlyeq & (x)(P_1^{nf} \mid P_2^{nf}) & \text{using the induction hypothesis} \\
 & = & (x)(\lceil \overline{x_1} \rceil P_1^{par} \mid \lceil \overline{x_2} \rceil P_2^{par}) & \text{by definition of thread normal form} \\
 & \preccurlyeq & \lceil \overline{y_1 y_2} \rceil (x)(P_1^{par} \mid P_2^{par}) & \text{by [SB-CAST-NEW],} \\
 & & & \text{[SB-CAST-SWAP] and [SB-PAR-COMM]} \\
 & = & P^{nf} & \text{by definition of } P^{nf}
\end{array}$$

Case [TB-CAST]. Then there exist x, Q, Γ', S and T such that

- $P = \lceil x \rceil Q$
- $\Gamma = \Gamma', x : S$
- $\Gamma', x : T \vdash_{\mathsf{ind}} Q$
- $S \leqslant T$

Using the induction hypothesis on $\Gamma', x : T \vdash_{\mathsf{ind}} Q$ we deduce that there exists Q^{nf} such that $Q \preccurlyeq Q^{nf}$. We conclude by taking $P^{nf} \stackrel{\mathsf{def}}{=} \lceil x \rceil Q^{nf}$ using the fact that $\preccurlyeq$ is a pre-congruence. □

In order to show that every well-typed, closed process in thread normal form can also be rewritten in proximity normal form we prove Lemma 4.3.9, which pushes a restriction (x) next to a process in which x occurs free, which might as well be an x-thread.

Lemma 4.3.9 (Proximity)**.** If $x \in \mathsf{fn}(P) \setminus \mathsf{bn}(\mathcal{C})$, then $(x)(\mathcal{C}[P] \mid Q) \preccurlyeq \mathcal{D}[(x)(P \mid Q)]$ for some $\mathcal{D}$.

Proof. By induction on the structure of $\mathcal{C}$ and by cases on its shape.

Case $\mathcal{C} = [\ ]$. We conclude by taking $\mathcal{D} \stackrel{\mathsf{def}}{=} [\ ]$ using the reflexivity of $\preccurlyeq$.

Case $\mathcal{C} = (y)(\mathcal{C}' \mid R)$. From the hypothesis $x \in \mathsf{fn}(P) \setminus \mathsf{bn}(\mathcal{C})$ we deduce $x \neq y$ and $x \in \mathsf{fn}(P) \setminus \mathsf{bn}(\mathcal{C}')$. Using the induction hypothesis we deduce that there exists $\mathcal{D}'$ such that $(x)(\mathcal{C}'[P]|Q) \preccurlyeq \mathcal{D}'[(x)(P|Q)]$. Take $\mathcal{D} \stackrel{\mathsf{def}}{=} (y)(\mathcal{D}'|R)$. We conclude

$$\begin{array}{rcll}
(x)(\mathcal{C}[P] \mid Q) & = & (x)((y)(\mathcal{C}'[P] \mid R) \mid Q) & \text{by definition of } \mathcal{C} \\
& \preccurlyeq & (x)(Q \mid (y)(\mathcal{C}'[P] \mid R)) & \text{by [SB-PAR-COMM]} \\
& \preccurlyeq & (y)((x)(Q \mid \mathcal{C}'[P]) \mid R) & \text{by [SB-PAR-ASSOC]} \\
& & & \text{and } x \in \mathsf{fn}(\mathcal{C}'[P]) \\
& \preccurlyeq & (y)((x)(\mathcal{C}'[P] \mid Q) \mid R) & \text{by [SB-PAR-COMM]} \\
& \preccurlyeq & (y)(\mathcal{D}'[(x)(P \mid Q)] \mid R) & \text{by induction hypothesis} \\
& = & \mathcal{D}[(x)(P \mid Q)] & \text{by definition of } \mathcal{D}
\end{array}$$

where, in using [SB-PAR-ASSOC], we note that $x \in \mathsf{fn}(\mathcal{C}'[P])$ since $x \in \mathsf{fn}(P) \setminus \mathsf{bn}(\mathcal{C})$.

Case $\mathcal{C} = (y)(R \mid \mathcal{C}')$. Symmetric of the previous case.

Case $\mathcal{C} = \lceil y \rceil \mathcal{C}'$ *and* $x \neq y$. Using the induction hypothesis we deduce that there exists $\mathcal{D}'$ such that $(x)(\mathcal{C}'[P]|Q) \preccurlyeq \mathcal{D}'[(x)(P|Q)]$. Take $\mathcal{D} \stackrel{\mathsf{def}}{=} \lceil y \rceil \mathcal{D}'$. We conclude

$$\begin{array}{rcll}
(x)(\mathcal{C}[P] \mid Q) & = & (x)(\lceil y \rceil \mathcal{C}'[P] \mid Q) & \text{by definition of } \mathcal{C} \\
& \preccurlyeq & \lceil y \rceil (x)(\mathcal{C}'[P] \mid Q) & \text{by [SB-CAST-SWAP] and } x \neq y \\
& \preccurlyeq & \lceil y \rceil \mathcal{D}'[(x)(P \mid Q)] & \text{using the induction hypothesis} \\
& = & \mathcal{D}[(x)(P \mid Q)] & \text{by definition of } \mathcal{D}
\end{array}$$

Case $\mathcal{C} = \lceil x \rceil \mathcal{C}'$. Using the induction hypothesis we deduce that there exists $\mathcal{D}$ such that $(x)(\mathcal{C}'[P] \mid Q) \preccurlyeq \mathcal{D}[(x)(P \mid Q)]$. We conclude

$$\begin{array}{rcll}
(x)(\mathcal{C}[P] \mid Q) & = & (x)(\lceil x \rceil \mathcal{C}'[P] \mid Q) & \text{by definition of } \mathcal{C} \\
& \preccurlyeq & (x)(\mathcal{C}'[P] \mid Q) & \text{by [SB-CAST-NEW]} \\
& \preccurlyeq & \mathcal{D}[(x)(P \mid Q)] & \text{using the induction hypothesis}
\end{array}$$ □

We can now prove the fact that every well-typed, closed process in thread normal form can be rewritten using structural pre-congruence either to done or to a process in proximity normal form. Note that this property the one that, combined with the *compatibility* of the involved session, guarantees deadlock freedom.

Lemma 4.3.10 (Quasi Deadlock Freedom)**.** If $\emptyset \vDash^{\mu} P^{nf}$, then $P^{nf} = \mathsf{done}$ or $P^{nf} \preccurlyeq Q^{nf}$ for some Q^{nf} in proximity normal form.

Proof. A simple induction on the derivation of $\emptyset \vDash^{\mu} P^{nf}$ allows us to deduce that P^{nf} consists of k sessions and $k+1$ threads. If $k = 0$, then we conclude $P^{nf} = \mathsf{done}$. If $k > 0$, then each of the $k+1$ threads is an x_i-thread for some x_i. Since there are $k+1$ threads but only k distinct sessions, it must be the case that $x_i = x_j$ for some $1 \leq i < j \leq k+1$. In other words, there exist $\mathcal{C}$, $\mathcal{C}_1$, $\mathcal{C}_2$, P_1^{th} and P_2^{th} such that P_1^{th} and P_2^{th} are x-threads and $P^{nf} = \mathcal{C}[(x)(\mathcal{C}_1[P_1^{th}] \mid \mathcal{C}_2[P_2^{th}])]$. We conclude

$$\begin{array}{rcll}
P^{nf} & = & \mathcal{C}[(x)(\mathcal{C}_1[P_1^{th}] \mid \mathcal{C}_2[P_2^{th}])] & \text{by definition of } P^{nf} \\
 & \preccurlyeq & \mathcal{C}[\mathcal{D}_1[(x)(P_1^{th} \mid \mathcal{C}_2[P_2^{th}])]] & \text{for some } \mathcal{D}_1 \text{ by Lemma 4.3.9} \\
 & \preccurlyeq & \mathcal{C}[\mathcal{D}_1[(x)(\mathcal{C}_2[P_2^{th}] \mid P_1^{th})]] & \text{by [\textsc{s-par-comm}]} \\
 & \preccurlyeq & \mathcal{C}[\mathcal{D}_1[\mathcal{D}_2[(x)(P_2^{th} \mid P_1^{th})]]] & \text{for some } \mathcal{D}_2 \text{ by Lemma 4.3.9} \\
 & \stackrel{\mathsf{def}}{=} & Q^{nf} &
\end{array}$$

where $x = x_i = x_j$. The fact that Q^{nf} is in thread normal form follows from the observation that P^{nf} does not have unguarded casts (it is a closed process in thread normal form) so the pre-congruence rules applied here and in Lemma 4.3.9 do not move casts around. We conclude that Q^{nf} is in proximity normal form by its shape. □

4.3.4 Soundness

𝄢 Here we prove the soundness of the type system (see Theorem 4.2.1). As already hinted at in Section 4.2.2, the proof is loosely based on the method of helpful directions [Francez, 1986], namely on the property that a (well-typed) process *may* reduce in such a way that its measure strictly decreases. Recall that this property is not true for every reduction. Here we assume that the reducing process is in proximity normal form. The same result will be generalized later on.

Lemma 4.3.11. If $\Gamma \vDash^{\mu} P^{nf}$ where P^{nf} is in proximity normal form, then there exist Q and $\nu < \mu$ such that $P^{nf} \Rightarrow^{+} Q$ and $\Gamma \vDash^{\nu} Q$.

Proof. From the hypothesis that P^{nf} is in proximity normal form we know that $P^{nf} = \mathcal{C}[(x)(P_1^{th} \mid P_2^{th})]$ for some $\mathcal{C}$, x and P_1^{th}, P_2^{th} x-threads. We reason by induction on $\mathcal{C}$ and by cases on its shape.

Case $\mathcal{C} = [\ ]$. From [mtb-thread] and [mtb-par] we deduce that there exist Γ_i, S_i, n_i for $i = 1, 2$ such that

- $\Gamma = \Gamma_1, \Gamma_2$
- $\mu = (n_1 + n_2, \|S_1, S_2\|)$
- $\Gamma_1, x : S_1 \vdash^{n_1} P_1^{th}$
- $\Gamma_2, x : S_2 \vdash^{n_2} P_2^{th}$
- $S_1 \sim S_2$

From the hypothesis that $S_1 \sim S_2$ we deduce $S_1 \xRightarrow{\overline{\varphi}} \overline{\pi}\mathsf{end}, S_2 \xRightarrow{\varphi} \pi\mathsf{end}$ for some φ. We now reason on the rank of the session and on the shape of S_i. For the sake of simplicity, we implicitly apply [SB-PAR-COMM] at process level.

If $\|S_1, S_2\| = 0$, then $S_1 = \overline{\pi}\mathsf{end}$ and $S_2 = \pi\mathsf{end}$ for some π.

- *Case* $S_1 = ?\mathsf{end}$ *and* $S_2 = !\mathsf{end}$. Then
  - $\Gamma_2 = \emptyset$ and $P_2^{th} = \mathsf{close}\, x$
  - $P_1^{th} = \mathsf{wait}\, x.Q$
  - $\Gamma_1 \vdash^{n_1} Q$

  From Lemma 4.3.3 we deduce that $\Gamma_1 \vDash^{\nu} Q$ for some $\nu \leq (n_1, 0)$. We conclude observing that $P^{nf} \to Q$ by [RB-SIGNAL] and that $\nu \leq (n_1, 0) < (n_1 + n_2, \|S_1, S_2\|) = \mu$.

If $\|S_1, S_2\| > 1$, then $S_1 \mid S_2 \xrightarrow{\tau} \dots \xrightarrow{\tau} \overline{\pi}\mathsf{end} \mid \pi\mathsf{end}$ first using [LB-TAU-L]/[LB-TAU-R] and [LB-SYNC]. Observe that [LB-PICK] is never used since we are considering the minimum reduction sequence; a synchronization through [LB-PICK] and [LB-SYNC] would lead to a longer reduction. T hen $S_1 \xrightarrow{!\mathsf{m}_k}$ and $S_2 \xrightarrow{?\mathsf{m}_k}$ for some m_k or $S_1 \xrightarrow{!U}$ and $S_2 \xrightarrow{?U}$ for some U.

- *Case* $S_1 = \sum_{i \in I} !\mathsf{m}_i.S_i'$ *and* $S_2 = \sum_{j \in J} ?\mathsf{m}_j.T_j$ *with* $k \in I$. From the hypothesis that $S_1 \sim S_2$ we deduce $I \subseteq J$. From [TB-TAG] we deduce that
  - $P_1^{th} = x!\{\mathsf{m}_i.P_i'\}_{i \in I}$
  - $P_2^{th} = x?\{\mathsf{m}_j.Q_j\}_{j \in J}$
  - $\Gamma_1, x : S_i' \vdash^{n_1} P_i'$ for all $i \in I$
  - $\Gamma_2, x : T_j \vdash^{n_2} Q_j$ for all $j \in J$

Let $Q \stackrel{\mathsf{def}}{=} (x)(P'_k|Q_k)$ and observe that $P^{nf} \Rightarrow^+ Q$ by [RB-PICK] and [RB-TAG]. From Lemma 5.3.4 we deduce that there exist $\mu_1 \leq (n_1, 0), \mu_2 \leq (n_2, 0)$ such that

– $x : S'_k \vDash^{\mu_1} P'_k$
– $x : T_k \vDash^{\mu_2} Q_k$

Let $\nu \stackrel{\mathsf{def}}{=} \mu_1 + \mu_2 + (0, \|S'_k, T_k\|)$. We conclude with one application of [MTB-PAR] observing that

$$\begin{array}{rcll} \nu & = & \mu_1 + \mu_2 + (0, \|S'_k, T_k\|) & \text{by def. of } \nu \\ & \leq & (n_1 + n_2, \|S'_k, T_k\|) & \text{by Lemma 4.3.3} \\ & < & (n_1 + n_2, \|S_1, S_2\|) & \text{before } \rightarrow \\ & = & \mu & \end{array}$$

- *Case* $S_1 = !S.T_1$ *and* $S_2 = ?S.T_2$.

From the hypothesis that $S_1 \sim S_2$ and Definition 2.3.1 we deduce $T_1 \sim T_2$ and from [TB-CHANNEL-OUT] and [TB-CHANNEL-IN] we deduce that

- $P_1^{th} = x!y.P'_1$
- $P_2^{th} = x?y.P'_2$
- $\Gamma_1, x : T_1 \vdash^{n_1} P'_1$
- $\Gamma_2, x : T_2, y : S \vdash^{n_2} P'_2$

Let $Q \stackrel{\mathsf{def}}{=} (x)(P'_1 \mid P'_2)$ and observe that $P^{nf} \rightarrow Q$ by [RB-CHANNEL]. From Lemma 4.3.3 we deduce that there exist $\mu_1 \leq (n_1, 0), \mu_2 \leq (n_2, 0)$ such that

- $\Gamma_1, x : T_1 \vDash^{\mu_1} P'_1$
- $\Gamma_2, x, y : S \vDash^{\mu_2} P'_2$

Let $\nu \stackrel{\mathsf{def}}{=} \mu_1 + \mu_2 + (0, \|T_1, T_2\|)$. We conclude with one application of [MTB-PAR] observing that

$$\begin{array}{rcll} \nu & = & \mu_1 + \mu_2 + (0, \|T_1, T_2\|) & \text{by definition of } \nu \\ & \leq & (n_1 + n_2, \|T_1, T_2\|) & \text{by Lemma 5.3.4} \\ & < & (n_1 + n_2, \|S_1, S_2\|) & \text{before reductions} \\ & = & \mu & \end{array}$$

Case $\mathcal{C} = (y)(\mathcal{D} \mid Q^{par})$. Let $R^{nf} \stackrel{\mathsf{def}}{=} \mathcal{D}[(x)(P_1^{th} \mid P_2^{th})]$ and observe that R^{nf} is in proximity normal form. From [MTB-PAR] we deduce that there exist Γ_i, S_i, μ_i for $i = 1, 2$ such that

- $\Gamma = \Gamma_1, \Gamma_2$
- $\Gamma_1, y : S_1 \vDash^{\mu_1} R^{nf}$
- $\Gamma_2, y : S_2 \vDash^{\mu_2} Q^{par}$
- $\mu = \mu_1 + \mu_2 + (0, \|S_1, S_2\|)$

Using the induction hypothesis on $\Gamma_1, y : S_1 \vDash^{\mu_1} R^{nf}$ we deduce that there exists Q' and $\nu' < \mu_1$ such that

- $R^{nf} \Rightarrow^+ Q'$
- $\Gamma_1, y : S_1 \vDash^{\nu'} Q'$

We conclude taking $Q \overset{\mathsf{def}}{=} (t)(Q' \mid Q^{par})$ and

$$\nu \overset{\mathsf{def}}{=} \nu' + \mu_2 + (0, \|S_1, S_2\|)$$

and observing that $\nu < \mu$ and $P^{nf} \Rightarrow^+ Q$ by [RB-PAR].

Case $\mathcal{C} = (x)(Q^{par} \mid \mathcal{D})$. Symmetric to the previous case.

Case $\mathcal{C} = \lceil y \rceil \mathcal{D}$. Observe that $x \neq y$. Let $R^{nf} \overset{\mathsf{def}}{=} \mathcal{D}[(x)(P_1^{th} \mid P_2^{th})]$ and note that R^{nf} is in proximity normal form. From [MTB-CAST] we deduce that there exists Δ, μ', S, T, m_y such that

- $\Gamma = \Delta, y : S$
- $S \leqslant_{m_y} T$
- $\mu = \mu' + (m_y, 0)$
- $\Delta, y : T \vDash^{\mu'} R^{nf}$

Using the induction hypothesis on $\Delta, y : T \vDash^{\mu'} R^{nf}$ we deduce that there exist Q' and $\nu' < \mu'$ such that $R^{nf} \Rightarrow^+ Q'$ and $\Delta, y : T \vDash^{\nu'} Q'$. We conclude taking $Q \overset{\mathsf{def}}{=} \lceil y \rceil Q'$ and $\nu \overset{\mathsf{def}}{=} \nu' + (m_y, 0)$ and observing that $\nu < \mu$ and $P^{nf} \Rightarrow^+ Q$ by [RB-CAST]. $\square$

Now we prove that any well-typed, closed process can be either rewritten to done using structural pre-congruence or reduced so as to obtain a strictly smaller measure.

Lemma 4.3.12. If $\emptyset \vDash^{\mu} P$, then either $P \preccurlyeq \mathsf{done}$ or $P \Rightarrow^+ Q$ and $\emptyset \vDash^{\nu} Q$ for some Q and $\nu < \mu$.

Proof. Using Lemma 4.3.7 we deduce that there exist P' in choice normal form such that $P \Rightarrow P'$ and $\emptyset \vDash^{\mu'} P'$ and $\mu' \leq \mu$. By Lemma 4.3.3 we deduce $\emptyset \vdash P'$. Using Lemma 4.3.8 we deduce that there exist P^{nf} such that $P' \preccurlyeq P^{nf}$. If $P^{nf} = \mathsf{done}$ there is nothing left to prove. If $P^{nf} \neq \mathsf{done}$, by Lemma 4.3.10 we deduce $P^{nf} \preccurlyeq Q^{nf}$ for some Q^{nf} in proximity normal form. From Lemma 4.3.4 we deduce $\emptyset \vDash^{\mu''} Q^{nf}$ for some $\mu'' \leq \mu'$. Using Lemma 4.3.11 we conclude that $Q^{nf} \Rightarrow^{+} Q$ and $\emptyset \vDash^{\nu} Q$ for some Q and $\nu < \mu'' \leq \mu' \leq \mu$. □

Using Lemma 4.3.12 we can prove that a well-typed, closed process weakly terminates. Namely, that there exists a finite reduction sequence to done.

Lemma 4.3.13 (Weak Termination)**.** If $\emptyset \vdash^{n} P$, then either $P \preccurlyeq \mathsf{done}$ or $P \Rightarrow^{+} \mathsf{done}$.

Proof. From Lemma 4.3.3 we deduce that there exists $\mu \leq (n, 0)$ such that $\emptyset \vDash^{\mu} P$. We proceed doing an induction on the lexicographically ordered pair μ. From Lemma 4.3.12 we deduce either $P \preccurlyeq \mathsf{done}$ or $P \Rightarrow^{+} Q$ and $\emptyset \vDash^{\nu} Q$ for some $\nu < \mu$. In the first case there is nothing left to prove. In the second case we use the induction hypothesis to deduce that either $Q \preccurlyeq \mathsf{done}$ or $Q \Rightarrow^{+} \mathsf{done}$. We conclude using either [RB-STRUCT] or the transitivity of $\Rightarrow^{+}$, respectively. □

The soundness of the type system is now a simple corollary.

Proof of Theorem 4.2.1. Consequence of Lemma 4.3.2 and Lemma 4.3.13. □

Chapter 5

Fair Termination Of Multiparty Sessions

In this chapter we generalize the type system in Chapter 4 to multiparty session types. In particular, we provide a type system for enforcing (successful) fair termination of multiparty sessions. In this case we refer to the notion of *coherence* (Definition 2.3.2). Again, we embed *fair subtyping* as the coherence-preserving relation. Notably, the *boundedness* properties mentioned in Section 4.2.1 are still required. For the sake of simplicity we do not analyze them step by step as the motivating examples can be straightforwardly adapted. Instead, we directly show how to enforce them.

Remark 5.0.1. The need of the type system that we present in this chapter comes from the fact the there exist multiparty sessions that cannot be decomposed in binary ones. Consider the following local types

$$\begin{array}{lcl} S_{\mathsf{p}} & : & \mathsf{q}!\{\mathsf{a}.\mathsf{r}!\mathsf{c}.!\mathsf{end},\ \mathsf{b}.!\mathsf{end}\} \\ S_{\mathsf{q}} & : & \mathsf{p}?\{\mathsf{a}.\mathsf{r}!\mathsf{ok}.!\mathsf{end},\ \mathsf{b}.\mathsf{r}!\mathsf{no}.!\mathsf{end}\} \\ S_{\mathsf{r}} & : & \mathsf{q}?\{\mathsf{ok}.\mathsf{p}?\mathsf{c}.?\mathsf{end},\ \mathsf{no}.?\mathsf{end}\} \end{array}$$

This example is paradigmatic because of the dependencies in the communications. Indeed, r receives c from p only if q receives a from p. Furthermore, the class of multiparty sessions that can be decomposed into linear ones consists of those sessions whose local types can be *projected* to all the participants appearing in them. (*i.e.* the *projections* exist). This is not true for the example above because the projections of S_{p} on r and of S_{r} on p are not defined as is such types the communication with r and p does not take place in all the branches. ⌟

The chapter is organized as follows. In Section 5.1 we present the syntax and the semantics of the calculus based on multiparty sessions. Section 5.2 shows the typing rules for such calculus. Differently from Section 4.2, we focus on some involved examples of processes instead of analyzing additional required properties (see Section 4.2.1). Then, in Section 5.3 we detail the soundness proof of the type system. At last, in Section 5.4 we compare the present type system, and consequently the one in Chapter 6, to existing works.

5.1 Calculus

In this section we define the calculus for multiparty sessions on which we apply our static analysis technique. The calculus is an extension of the one presented by Ciccone and Padovani [2022c] to multiparty sessions in the style of Scalas and Yoshida [2019] and that has been presented in Ciccone et al. [2022]. We recall some basic notions. We use an infinite set of *variables* ranged over by x, y, z, an infinite set of *session names* ranged over by s and t, a set of *roles* ranged over by p, q, r, a set of *message tags* ranged over by m, and a set of *process names* ranged over by A, B, C. As in the binary case, the different terminology for labels is needed to avoid confusion with the labels in Section 1.3. We use roles to distinguish the participants of a session. In particular, an *endpoint* s[p] consists of a session name s and a role p and is used by the participant with role p to interact with the other participants of the session s. We use u and v to range over *channels*, which are either variables or session endpoints. We write $\overline{x}$ and $\overline{u}$ to denote possibly empty sequences of variables and channels, extending this notation to other entities. We use π to range over the elements of the set $\{?, !\}$ of *polarities*, distinguishing input actions (?) from output actions (!).

5.1.1 Syntax of Processes

A *program* is a finite set of *definitions* of the form $A(\overline{x}) \triangleq P$, at most one for each process name, where P is a term generated by the syntax shown in Figure 5.1. The term done denotes the terminated process that performs no action. The term $A\langle\overline{u}\rangle$ denotes the invocation of the process

P, Q, R	$::=$		**Process**			
		done	termination	$\mid$	$A\langle\overline{u}\rangle$	invocation
	$\mid$	$\mathsf{wait}\, u.P$	signal in	$\mid$	$\mathsf{close}\, u$	signal out
	$\mid$	$u[\mathsf{p}]?(x).P$	channel in	$\mid$	$u[\mathsf{p}]!v.P$	channel out
	$\mid$	$u[\mathsf{p}]\pi\{\mathsf{m}_i.P_i\}_{i\in I}$	tag in/out	$\mid$	$P \oplus Q$	choice
	$\mid$	$(s)(P_1 \mid \cdots \mid P_n)$	session	$\mid$	$\lceil u\rceil P$	cast

Figure 5.1: Syntax of processes

with name A passing the channels $\overline{u}$ as arguments. When $\overline{u}$ is empty we just write A instead of $A\langle\rangle$. The term $\mathsf{close}\, u$ denotes the process that sends a termination signal on the channel u, whereas $\mathsf{wait}\, u.P$ denotes the process that waits for a termination signal from channel u and then continues as P. The term $u[\mathsf{p}]!v.P$ denotes the process that sends the channel v on the channel u to the role p and then continues as P. Dually, $u[\mathsf{p}]?(x).P$ denotes the process that receives a channel from the role p on the channel u and then continues as P where x is replaced with the received channel. The term $u[\mathsf{p}]\pi\{\mathsf{m}_i.P_i\}_{i\in I}$ denotes a process that exchanges one of the tags m_i on the channel u with the role p and then continues as P_i. Whether the tag is sent or received depends on the polarity π and, as it will be clear from the operational semantics, the polarity π also determines whether the process behaves as an internal choice (when π is $!$) or an external choice (when π is $?$). In the first case the process chooses *actively* the tag being sent, whereas in the second case the process reacts *passively* to the tag being received. We assume that I is finite and non-empty and also that the tags m_i are pairwise distinct. For brevity, we write $u[\mathsf{p}]\pi\mathsf{m}_k.P_k$ instead of $u[\mathsf{p}]\pi\{\mathsf{m}_i.P_i\}_{i\in I}$ when I is the singleton set $\{k\}$. The term $P \oplus Q$ denotes a process that non-deterministically behaves either as P or as Q.

A term $(s)(P_1 \mid \cdots \mid P_n)$ with $n \geq 1$ denotes the parallel composition of n processes, each of them being a participant of the session s. Each process is associated with a distinct a role p_i and communicates in s through the endpoint $s[\mathsf{p}_i]$. Combining session creation and parallel composition in a single form is common in session type systems based on linear logic [Caires et al., 2016, Wadler, 2014, Lindley and Morris, 2016] and helps guaranteeing deadlock freedom. Finally, a *cast* $\lceil u\rceil P$ denotes a process that behaves exactly as P. This form is only relevant for the type system and denotes the fact that the type of u is subject to an application of subtyping.

The free and bound names of a process are defined as usual, the latter ones being easily recognizable as they occur within round parenteses. We

[SM-PAR-COMM]	$(s)(\overline{P} \mid P \mid Q \mid \overline{Q}) \preccurlyeq (s)(\overline{P} \mid Q \mid P \mid \overline{Q})$	
[SM-PAR-ASSOC]	$(s)(\overline{P} \mid (t)(R \mid \overline{Q})) \preccurlyeq (t)((s)(\overline{P} \mid R) \mid \overline{Q})$	if $s \in \mathsf{fn}(R)$ and $t \notin \mathsf{fn}(P)$, $\forall Q.s \notin \mathsf{fn}(Q)$
[SM-CAST-COMM]	$\lceil u \rceil \lceil v \rceil P \preccurlyeq \lceil v \rceil \lceil u \rceil P$	
[SM-CAST-NEW]	$(s)(\lceil s[\mathsf{p}] \rceil P \mid \overline{Q}) \preccurlyeq (s)(P \mid \overline{Q})$	
[SM-CAST-SWAP]	$(s)(\lceil t[\mathsf{p}] \rceil P \mid \overline{Q}) \preccurlyeq \lceil t[\mathsf{p}] \rceil (s)(P \mid \overline{Q})$	if $s \neq t$
[SM-CALL]	$A\langle \overline{u} \rangle \preccurlyeq P\{\overline{u}/\overline{x}\}$	if $A(\overline{x}) \stackrel{\triangle}{=} P$

Figure 5.2: Structural precongruence of processes

write $\mathsf{fn}(P)$ for the set of free names of P and we identify processes modulo renaming of bound names. Note that $\mathsf{fn}(P)$ may contain variables and session names, but not endpoints. Occasionally we write $A(\overline{x}) \stackrel{\triangle}{=} P$ as a predicate or side condition, meaning that P is the process associated with the process name A. For each of such definitions we assume that $\mathsf{fn}(P) \subseteq \{\overline{x}\}$.

5.1.2 Operational Semantics

The operational semantics of processes is given by the structural precongruence relation $\preccurlyeq$ defined in Figure 5.2 and the reduction relation $\rightarrow$ defined in Figure 5.3. As usual, structural precongruence allows us to rearrange the structure of processes without altering their meaning, whereas reduction expresses an actual computation or interaction step. The adoption of a structural *pre*congruence (as opposed to a more common congruence relation) is not strictly necessary, but it simplifies the technical development by reducing the number of cases we have to consider in proofs without affecting the properties of the calculus in any way (see Remark 4.1.1).

Rules [SM-PAR-COMM] and [SM-PAR-ASSOC] state commutativity and associativity of parallel composition of processes (we write $\overline{P}$ to denote possibly empty parallel compositions of processes). In [SM-PAR-ASSOC], the side condition $s \in \mathsf{fn}(R)$ makes sure that R is indeed a participant of the session s. We write $\forall Q.s \notin \mathsf{fn}(Q)$ to state that s is not free in each of the processes $\overline{Q}$. Moreover, note that this rule only states right-to-left associativity. Left-to-right associativity is derivable from this rule and repeated uses of [SM-PAR-COMM]. Rule [SM-CAST-COMM] allows us to swap two consecutive casts. Rule [SM-CAST-NEW] removes an unguarded cast on an endpoint of the restricted session (we refer to this operation as "performing the cast"). Rule [SM-CAST-SWAP] swaps a cast and a restricted session as long as the

RM-CHOICE
$$\frac{}{P_1 \oplus P_2 \rightarrow P_k}\ k \in \{1,2\}$$

RM-SIGNAL
$$\frac{}{(s)(\mathsf{wait}\, s[\mathsf{p}].P \mid \mathsf{close}\, s[\mathsf{q}_1] \mid \cdots \mid \mathsf{close}\, s[\mathsf{q}_n]) \rightarrow P}$$

RM-CHANNEL
$$\frac{}{(s)(s[\mathsf{p}][\mathsf{q}]!v.P \mid s[\mathsf{q}][\mathsf{p}]?(x).Q \mid \overline{R}) \rightarrow (s)(P \mid Q\{v/x\} \mid \overline{R})}$$

RM-PICK
$$\frac{}{(s)(s[\mathsf{p}][\mathsf{q}]!\{\mathsf{m}_i.P_i\}_{i\in I} \mid \overline{Q}) \rightarrow (s)(s[\mathsf{p}][\mathsf{q}]!\mathsf{m}_k.P_k \mid \overline{Q})}\ k \in I$$

RM-TAG
$$\frac{}{(s)(s[\mathsf{p}][\mathsf{q}]!\mathsf{m}_k.P \mid s[\mathsf{q}][\mathsf{p}]?\{\mathsf{m}_i.Q_i\}_{i\in I} \mid \overline{R}) \rightarrow (s)(P \mid Q_k \mid \overline{R})}\ k \in I$$

RM-PAR
$$\frac{P \rightarrow Q}{(s)(P \mid \overline{R}) \rightarrow (s)(Q \mid \overline{R})}$$

RM-CAST
$$\frac{P \rightarrow Q}{\lceil u \rceil P \rightarrow \lceil u \rceil Q}$$

RM-STRUCT
$$\frac{P \preccurlyeq P' \qquad P' \rightarrow Q' \qquad Q' \preccurlyeq Q}{P \rightarrow Q}$$

Figure 5.3: Reduction of processes

endpoint in the cast refers to a different session. Finally, rule [SM-CALL] unfolds a process invocation to its definition. Hereafter, we write $\{u/x\}$ for the capture-avoiding substitution of each free occurrence of x with u and $\{\overline{u}/\overline{x}\}$ for its natural extension to equal-length tuples of variables and names. The rules [SM-CAST-NEW], [SM-CAST-SWAP] and [SM-CALL] are not invertible: by [SM-CAST-NEW] casts can only be removed but never added; by [SM-CAST-SWAP] casts can only be moved closer to their restriction, so that they can be eventually performed by [SM-CAST-NEW]; by [SM-CALL] process invocations can only be unfolded.

The reduction relation is quite standard. Rule [RM-CHOICE] reduces $P_1 \oplus P_2$ to either P_1 or P_2, non deterministically. Rule [RM-SIGNAL] terminates a session in which all participants ($\mathsf{q}_1, \ldots, \mathsf{q}_n$) but one ($\mathsf{p}$) are sending a termination signal and p is waiting for it; the resulting process is the continuation of the participant p. Rule [RM-CHANNEL] models the exchange of a channel among two participants of a session. Rule [RM-PICK] models an internal choice whereby a process picks one particular tag m_k to send on a session. Rule [RM-TAG] synchronizes two participants p and q on the tag chosen by p. Finally, rules [RM-PAR], [RM-CAST] and [RM-STRUCT] close reductions under parallel compositions and casts and by structural precongruence.

5.1.3 Examples

In the rest of this section we illustrate the main features of the calculus with some examples. For none of them the existing multiparty session type systems are able to guarantee progress. First of all, we formally define in our calculus the processes corresponding to (a slightly different) **buyer**, **seller** and **carrier** from Example 2.1.1 that was only informally presented.

Example 5.1.1 (Buyer - Seller - Carrier)**.** *Consider the following definitions:*

$$
\begin{aligned}
\mathit{Main} &\stackrel{\triangle}{=} (s)(\mathit{Buyer}\langle s[\mathsf{buyer}]\rangle \mid \mathit{Seller}\langle s[\mathsf{seller}]\rangle \mid \mathit{Carrier}\langle s[\mathsf{carrier}]\rangle)\\
\mathit{Buyer}(x) &\stackrel{\triangle}{=} x[\mathsf{seller}]!\{\mathsf{add}.x[\mathsf{seller}]!\mathsf{add}.\mathit{Buyer}\langle x\rangle, \mathsf{pay}.\mathsf{close}\,x\}\\
\mathit{Seller}(x) &\stackrel{\triangle}{=} x[\mathsf{buyer}]?\{\mathsf{add}.\mathit{Seller}\langle x\rangle, \mathsf{pay}.x[\mathsf{carrier}]!\mathsf{ship}.\mathsf{close}\,x\}\\
\mathit{Carrier}(x) &\stackrel{\triangle}{=} x[\mathsf{seller}]?\mathsf{ship}.\mathsf{wait}\,x.\mathsf{done}
\end{aligned}
$$

Note that the buyer either sends pay *or it sends two* add *messages in a row before repeating this behavior. That is, this particular buyer always adds an even number of items to the shopping cart. Nonetheless, the buyer periodically has a chance to send a* pay *message and terminate. Therefore, the execution of the program in which the buyer only sends* add *is unfair according to Definition 2.2.3 hence this program is fairly terminating.* ⌟

Example 5.1.2 (Purchase with negotiation)**.** *Consider a variation of Example 2.1.1 in which the buyer, before making the payment, negotiates with a secondary buyer for an arbitrarily long time. The interaction happens in two nested sessions, an outer one involving the primary buyer, the seller and the carrier, and an inner one involving only the two buyers. We model the interaction as the program below, in which we collapse role names to their* *initials.*

$$
\begin{aligned}
\mathit{Main} &\triangleq (s)(\mathit{Buyer}\langle s[\mathsf{b}]\rangle \mid \mathit{Seller}\langle s[\mathsf{s}]\rangle \mid \mathit{Carrier}\langle s[\mathsf{c}]\rangle)\\
\mathit{Buyer}(x) &\triangleq x[\mathsf{s}]!\mathsf{query}.x[\mathsf{s}]?\mathsf{price}.(t)(\mathit{Buyer}_1\langle x, t[\mathsf{b}_1]\rangle \mid \mathit{Buyer}_2\langle t[\mathsf{b}_2]\rangle)\\
\mathit{Seller}(x) &\triangleq x[\mathsf{b}]?\mathsf{query}.x[\mathsf{b}]!\mathsf{price}.x[\mathsf{b}]?\{\mathsf{pay}.x[\mathsf{c}]!\mathsf{ship}.\mathsf{close}\,x,\\
&\qquad\qquad \mathsf{cancel}.x[\mathsf{c}]!\mathsf{cancel}.\mathsf{close}\,x\}\\
\mathit{Carrier}(x) &\triangleq x[\mathsf{s}]?\{\mathsf{ship}.x[\mathsf{b}]!\mathsf{box}.\mathsf{close}\,x, \mathsf{cancel}.\mathsf{close}\,x\}\\
\mathit{Buyer}_1(x,y) &\triangleq y[\mathsf{b}_2]!\{\mathsf{split}.y[\mathsf{b}_2]?\{\mathsf{yes}.\lceil x\rceil x[\mathsf{s}]!\mathsf{ok}.x[\mathsf{c}]?\mathsf{box}.\mathsf{wait}\,x.\mathsf{wait}\,y.\mathsf{done},\\
&\qquad\qquad \mathsf{no}.\mathit{Buyer}_1\langle x,y\rangle\},\\
&\qquad \mathsf{giveup}.\mathsf{wait}\,y.\lceil x\rceil x[\mathsf{s}]!\mathsf{cancel}.\mathsf{wait}\,x.\mathsf{done}\}\\
\mathit{Buyer}_2(y) &\triangleq y[\mathsf{b}_1]?\{\mathsf{split}.y[\mathsf{b}_1]!\{\mathsf{yes}.\mathsf{close}\,y, \mathsf{no}.\mathit{Buyer}_2\langle y\rangle\}, \mathsf{giveup}.\mathsf{close}\,y\}
\end{aligned}
$$

The buyer queries the seller which replies with a price. At this point, Buyer creates a new session t and forks as a primary buyer Buyer_1 and a secondary buyer Buyer_2. The interaction between the two sub-buyers goes on until either Buyer_1 gives up or Buyer_2 accepts its share of the price. In the former case, the primary buyer waits for the internal session to terminate and cancel*s the order with the seller which, in turn, aborts the transaction with the carrier. In the latter case, the buyer confirms the order to the seller, which then instructs the carrier to* ship *a* box *to the buyer.*

Note that the outermost session s, taken in isolation, terminates in a bounded number of interactions, but its progress cannot be established without assuming that the innermost session t terminates. In particular, if the two buyers keep negotiating forever, the seller and the carrier starve. However, the innermost session can terminate if Buyer_1 sends giveup *to Buyer_2 or if Buyer_2 sends* yes *to Buyer_1. Thus, the run in which the two buyers negotiate forever is unfair, the session t fairly terminates and the session s terminates as well.*

On the technical side, note that the definition of Buyer_1 contains two casts on the variable x. As we will see in example 5.2.2, these casts are necessary for the typeability of Buyer_1 to account for the fact that x is used differently *in two distinct branches of the process.* ⌟

Example 5.1.3 (Parallel merge sort)**.** *To illustrate an example of program that creates an unbounded number of sessions we model a parallel version of* *the merge sort algorithm.*

$$\begin{aligned}
\mathit{Main} &\stackrel{\triangle}{=} (s)(s[\mathsf{m}][\mathsf{w}]!\mathsf{req}.s[\mathsf{m}][\mathsf{w}]?\mathsf{res}.\mathsf{wait}\,s.\mathsf{done} \mid \mathit{Sort}\langle s[\mathsf{w}]\rangle)\\
\mathit{Sort}(x) &\stackrel{\triangle}{=} x[\mathsf{m}]?\mathsf{req}.\\
&\qquad ((t)(\mathit{Merge}\langle x, t[\mathsf{m}]\rangle \mid \mathit{Sort}\langle t[\mathsf{w}_1]\rangle \mid \mathit{Sort}\langle t[\mathsf{w}_2]\rangle) \oplus x[\mathsf{m}]!\mathsf{res}.\mathsf{close}\,x)\\
\mathit{Merge}(x, y) &\stackrel{\triangle}{=} y[\mathsf{w}_1]!\mathsf{req}.y[\mathsf{w}_2]!\mathsf{req}.y[\mathsf{w}_1]?\mathsf{res}.y[\mathsf{w}_2]?\mathsf{res}.\mathsf{wait}\,y.x[\mathsf{m}]!\mathsf{res}.\mathsf{close}\,x
\end{aligned}$$

The program starts as a single session s in which a master m sends the initial collection of data to the worker w as a req message and waits for the result. The worker is modeled as a process Sort that decides whether to sort the data by itself (right branch of the choice in Sort), in which case it sends the result directly to the master, or to partition the collection (left branch of the choice in Sort). In the latter case, it creates a new session t in which it sends requests to two sub-workers w_1 and w_2, it gathers the partial results from them and gets back to the master with the complete result.

Since a worker may always choose to start two sub-workers in a new session, the number of sessions that may be created by this program is unbounded. At the same time, each worker may also choose to complete its task without creating new sessions. So, while in principle there exists a run of this program that keeps creating new sessions forever, this run is unfair according to Definition 2.2.3. ⌟

5.2 Type System

In this section we describe the type system for the calculus of multiparty sessions of Section 5.1. The typing judgments have the form $\Gamma \vdash^n P$, meaning that the process P is well typed in the typing context Γ and has rank n. As usual, the *typing context* is a map associating channels with session types (Section 1.3.2) and is meant to contain an association for each name in $\mathsf{fn}(P)$. We write $u_1 : S_1, \dots, u_n : S_n$ for the map with domain $\{u_1, \dots, u_n\}$ that associates u_i with S_i. Occasionally we write $\overline{u : S}$ for the same context, when the number and the specific associations are unimportant. We also assume that endpoints occurring in a typing context have different session names. That is, $s[\mathsf{p}], s[\mathsf{q}] \in \mathsf{dom}(\Gamma)$ implies $\mathsf{p} = \mathsf{q}$. This constraint makes sure that each well-typed process plays exactly one role in each of the sessions in which it participates. It is also a common assumption made in all multiparty session calculi. We use Γ and Δ to range over typing contexts, we write $\emptyset$ for the empty context and Γ, Δ for the union of Γ and Δ when they have disjoint domains and disjoint sets of session names. The

$$\text{TM-DONE}\quad \frac{}{\emptyset \vdash^{n} \mathsf{done}} \qquad \text{TM-CALL}\quad \frac{\overline{u : S} \vdash^{n} P\{\overline{u}/\overline{x}\}}{\overline{u : S} \vdash^{n+m} A\langle\overline{u}\rangle}\ A : [\overline{S}; n], A(\overline{x}) \triangleq P$$

$$\text{TM-WAIT}\quad \frac{\Gamma \vdash^{n} P}{\Gamma, u : ?\mathsf{end} \vdash^{n} \mathsf{wait}\, u.P} \qquad \text{TM-CLOSE}\quad \frac{}{u : !\mathsf{end} \vdash^{n} \mathsf{close}\, u}$$

$$\text{TM-CHANNEL-IN}\quad \frac{\Gamma, u : T, x : S \vdash^{n} P}{\Gamma, u : \mathsf{p}?S.T \vdash^{n} u[\mathsf{p}]?(x).P} \qquad \text{TM-CHANNEL-OUT}\quad \frac{\Gamma, u : T \vdash^{n} P}{\Gamma, u : \mathsf{p}!S.T, v : S \vdash^{n} u[\mathsf{p}]!v.P}$$

$$\text{TM-TAG}\quad \frac{\forall i \in I : \Gamma, u : S_i \vdash^{n} P_i}{\Gamma, u : \sum_{i\in I} \mathsf{p}\pi\mathsf{m}_i.S_i \vdash^{n} u[\mathsf{p}]\pi\{\mathsf{m}_i.P_i\}_{i\in I}}$$

$$\text{TM-CHOICE}\quad \frac{\Gamma \vdash^{n_1} P_1 \qquad \Gamma \vdash^{n_2} P_2}{\Gamma \vdash^{n_k} P_1 + P_2}\ k \in \{1, 2\} \qquad \text{TM-CAST}\quad \frac{\Gamma, u : T \vdash^{n} P}{\Gamma, u : S \vdash^{m+n} \lceil u \rceil P}\ S \leqslant_m T$$

$$\text{TM-PAR}\quad \frac{\forall i \in \{1, \dots, h\} : \Gamma_i, s[\mathsf{p}_i] : S_i \vdash^{n_i} P_i}{\Gamma_1, \dots, \Gamma_h \vdash^{1+n_1+\cdots+n_h} (s)(P_1 \mid \cdots \mid P_h)}\ \#\{\mathsf{p}_i \triangleright S_i\}_{i=1..h}$$

$$\text{COM-TAG}\quad \frac{\Gamma, u : S_k \vdash^{n} P_k}{\Gamma, u : \sum_{i\in I} \mathsf{p}\pi\mathsf{m}_i.S_i \vdash^{n} u[\mathsf{p}]\pi\{\mathsf{m}_i.P_i\}_{i\in I}}\ k \in I \qquad \text{COM-CHOICE}\quad \frac{\Gamma \vdash^{n} P_k}{\Gamma \vdash^{n} P_1 \oplus P_2}\ k \in \{1, 2\}$$

Figure 5.4: Typing rules

rank n in a typing judgment estimates the number of sessions that P has to create and the number of casts that P has to perform in order to terminate. The fact that the rank is finite suggests that so is the effort required by P to terminate.

5.2.1 Typing Rules

The typing rules are shown in Figure 5.4 as a *generalized inference system* (see Section 1.2) in which, we recall, the singly-lined rules are interpreted coinductively and the called *corules* are interpreted inductively.

For more details about the interpretation, we refer to Section 1.2. We type check a program $\{A_i(\overline{x_i}) \triangleq P_i\}_{i\in I}$ under a global set of assignments $\{A_i : [\overline{S_i}; n_i]\}_{i\in I}$ associating each process name A_i with a tuple of session types $\overline{S_i}$, one for each of the variables in $\overline{x_i}$, and a rank n_i. The program is well typed if $\overline{x_i : S_i} \vdash^{n_i} P_i$ is derivable for every $i \in I$, establishing that the tuple $\overline{S_i}$ corresponds to the way the variables $\overline{x_i}$ are used by P_i and that n_i is a feasible rank annotation for P_i. We now describe the typing rules in detail.

The rule [TM-DONE] states that the terminated process is well typed in the empty context, to make sure that no unused channels are left behind. Note that done can be given any rank, since it performs no casts and it creates no new sessions. The rule [TM-CALL] checks that a process invocation $A\langle\overline{u}\rangle$ is well typed by unfolding A into the process associated with A. The types associated with $\overline{u}$ must match those of the global assignment $A : [\overline{S}; n]$ and the rank of the process must be no greater than that of the invocation. The potential mismatch between the two ranks improves typeability in some corner cases. The rules [TM-WAIT] and [TM-CLOSE] concern processes that exchange termination signals. The channel being closed is consumed and, in the case of [TM-WAIT], no longer available in the continuation P. Again, $\mathsf{close}\, u$ can be typed with any rank whereas the rank of $\mathsf{wait}\, u.P$ coincides with that of P. The rules [TM-CHANNEL-IN] and [TM-CHANNEL-OUT] deal with the exchange of channels in a quite standard way. As in Section 4.2.2, note that the actual type of the exchanged channel is required to coincide with the expected one. In particular, no covariance or contravariance of input and output respectively is allowed. Relaxing the typing rule in this way would introduce implicit applications of subtyping that may compromise fair termination (see Example 4.2.11). In our type system, each application of subtyping must be explicitly accounted for as we will see when discussing [TM-CAST]. Rule [TM-TAG] deals with the exchange of tags. Channels that are not used for such communication must be used in the same way in all branches, whereas the type of the channel on which the message is exchanged changes accordingly. All branches are required to have the same rank, which also corresponds to the rank of the process. Unlike other presentations of this typing rule [Gay and Hole, 2005], we require the branches in the process to be matched exactly by those in the type. Again, this is to avoid implicit application of subtyping, which might jeopardize fair termination. The rule [TM-CHOICE] deals with non-deterministic choices and requires both continuations to be well typed in the same typing context. The judgment in the conclusion inherits the rank of one of the processes, typically the one with minimum rank. As we will see in Example 5.2.5, this makes it possible to model finite-rank processes that may create an unbounded number of sessions or that perform an unbounded number of casts.

The rule [TM-CAST] models the substitution principle induced by fair subtyping: when $S \leqslant_m T$ (Figure 3.3), a channel of type S can be used where a channel of type T is expected or, in dual fashion [Gay, 2016], a process using u according to T can be used in place of a process using u according to S. To keep track of this cast, the rank in the conclusion is augmented by the weight m of the subtyping relation between S and T. Note that the typing rule guesses the target type of the cast.

Finally, the rule [TM-PAR] deals with session creation and parallel composition. This rule is inspired to the *multiparty cut* rule found in linear logic interpretations of multiparty session types [Carbone et al., 2016, 2017] and provides a straightforward way for enforcing deadlock freedom. Each process in the composition must be well typed in a slice of the typing context augmented with the endpoint corresponding to its role. The session map of the new session must be coherent (Definition 2.3.2), implying that it fairly terminates. The rank of the composition is one plus the aggregated rank of the composed processes, to account for the fact that one more session has been created. Recall that coherence is a property expressed on the LTS of session maps (Definition 2.3.2) in line with the approach of Scalas and Yoshida [2019].

The typing rules described so far are interpreted *coinductively*. That is, in order for a rank n process P to be well typed in Γ there must be a *possibly infinite* derivation tree built with these rules and whose conclusion is the judgment $\Gamma \vdash^n P$. But in a generalized inference system like the one we are defining, this is not enough to establish that P is well typed. In addition, it must be possible to find *finite* derivation trees for all of the judgments occurring in this possibly infinite derivation tree using the discussed rules *and possibly* the corules, which we are about to describe. Since the additional derivation trees must be finite, all of their branches must end up with an application of [TM-DONE] or [TM-CLOSE], which are the only axioms in Figure 5.4 corresponding to the only terminated processes in Figure 5.1. So, the purpose of these finite typing derivations is to make sure that in every well-typed (sub-)process there exists a path that leads to termination. On the one hand, this is a sensible condition to require as our type system is meant to enforce fair process termination. On the other hand, insisting that these finite derivations can be built using only the typing rules discusses thus far is overly restrictive, for a process might have *one* path that leads to termination, but also alternative paths that lead to (recursive) process invocations. In fact, all of the processes we have discussed in Examples 5.1.1 to 5.1.3 are structured like this. The two corules

[COM-CHOICE] and [COM-TAG] in Figure 5.4 establish that, whenever a multi-branch process is dealt with, it suffices for *one* of the branches to lead to termination. A key detail to note in the case of [COM-CHOICE] is that the rank of the non-deterministic choice coincides with that of the branch that leads to termination. This makes sense recalling that the rank associated with a process represents the overall effort required for that process to terminate.

Let us recap the notion of well-typed process resulting from the typing rules of Figure 5.4.

Definition 5.2.1 (Well-typed process)**.** We say that P is *well typed* in the context Γ and has rank n if

1. There exists an arbitrary (possibly infinite) derivation tree obtained using the rules in Figure 5.4 and whose conclusion is $\Gamma \vdash^n P$
2. For each judgment in such tree there is a finite derivation obtained using the rules and the corules

Now we can prove a strong soundness result for our type system, stating that well-typed, closed processes can always successfully terminate no matter how they reduce. We analyze the proof in details in Section 5.3.

Theorem 5.2.1 (Soundness)**.** *If $\emptyset \vdash^n P$ and $P \Rightarrow Q$, then $Q \Rightarrow\preccurlyeq \mathsf{done}$.*

The implications of Theorem 4.2.1 that we presented in Section 4.2 still hold. Notably, the proof schema of Theorem 5.2.1 follows that of Theorem 4.2.1.

5.2.2 Examples

We dedicate the rest of Section 5.2 to the analysis of some examples that integrate all the features of the presented type system. We start from some basic examples and then we move to more involved ones. First, in Example 5.2.1 we take into account our slightly different variant of the running example (Example 5.1.1). For what concerns the problematic processes in Section 4.2.1, they are still valid in the multiparty context (see Remark 5.2.1). We use the rest of the examples to deal with the processes introduced in Section 5.1.3.

Remark 5.2.1 (Boundedness). All the problematic processes that we presented in Section 4.2.1 are still valid in the multiparty scenario and can be dealt with using the techniques that we mentioned for the binary case. In particular

Action-boundedness. Type system with corules.

$$A \triangleq A \qquad B \triangleq B \oplus B \qquad C \triangleq C \oplus \mathsf{done}$$

Session-boundedness. Rule [TM-PAR] increases the rank by one.

$$A \triangleq (s)(s[\mathsf{p}][\mathsf{q}]!\{\mathsf{a}.\mathsf{close}\,s[\mathsf{p}], \mathsf{b}.\mathsf{wait}\,s[\mathsf{p}].A\} \mid s[\mathsf{q}][\mathsf{p}]?\{\mathsf{a}.\mathsf{wait}\,s[\mathsf{q}].A, \mathsf{b}.\mathsf{close}\,s[\mathsf{q}]\})$$

Cast-boundedness. Rule [TM-CAST] increases the rank by the weight of the subtyping being applied.

$$B(x) \triangleq \lceil x \rceil x[\mathsf{seller}]!\mathsf{add}.B\langle x\rangle$$

⌟

Example 5.2.1. *Let us show some typing derivations for fragments of Example 5.1.1. Let S_b, S_s and S_c be the types from Example 2.3.2. We collapse roles to their initials. Let $S'_b = \mathsf{s}!\mathsf{add}\mathsf{s}!\mathsf{add}.S'_b + \mathsf{s}!\mathsf{pay}.!\mathsf{end}$. Concerning Buyer, we obtain the infinite derivation*

$$\dfrac{\dfrac{\dfrac{\vdots}{x : S'_b \vdash^0 \mathit{Buyer}\langle x\rangle}\,[\textsc{tm-call}]}{x : \mathsf{s}!\mathsf{add}.S'_b \vdash^0 x[\mathsf{s}]!\mathsf{add}.\mathit{Buyer}\langle x\rangle}\,[\textsc{tm-tag}] \qquad \dfrac{}{x : !\mathsf{end} \vdash^0 \mathsf{close}\,x}\,[\textsc{tm-close}]}{x : S'_b \vdash^0 x[\mathsf{s}]!\{\mathsf{add}.x[\mathsf{s}]!\mathsf{add}.\mathit{Buyer}\langle x\rangle, \mathsf{pay}.\mathsf{close}\,x\}}\,[\textsc{tm-tag}]$$

and, for each judgment in it, it is easy to find a finite derivation possibly using [COM-TAG]*. Concerning Main we obtain*

$$\dfrac{\dfrac{\vdots}{s[\mathsf{b}] : S'_b \vdash^0 \mathit{Buyer}\langle s[\mathsf{b}]\rangle}\,[\textsc{tm-call}] \qquad \dfrac{\vdots}{s[\mathsf{s}] : S_s \vdash^0 \mathit{Seller}\langle s[\mathsf{s}]\rangle} \;\vdots}{\emptyset \vdash^1 (s)(\mathit{Buyer}\langle s[\mathsf{b}]\rangle \mid \mathit{Seller}\langle s[\mathsf{s}]\rangle \mid \mathit{Carrier}\langle s[\mathsf{c}]\rangle)}\,[\textsc{tm-par}]$$

where the application of [TM-PAR] *is justified by the fact that $\mathsf{b} \triangleright S'_b \mid \mathsf{s} \triangleright S_s \mid \mathsf{c} \triangleright S_c$ is coherent. We recall that $S_b \leqslant_1 S'_b$ (Example 3.2.1). No participant creates new sessions or performs casts, so they all have zero rank. The rank of Main is 1 since it creates the session s.* ⌟

Example 5.2.2. *In this example we show that the process Buyer_1 playing the role b_1 in the inner session of Example 5.1.2 is well typed. For clarity, we recall its definition here:*

$$\begin{aligned}\mathit{Buyer}_1(x, y) \triangleq y[\mathsf{b}_2]!\{&\mathsf{split}.y[\mathsf{b}_2]?\{\mathsf{yes}.\lceil x \rceil x[\mathsf{s}]!\mathsf{ok}.x[\mathsf{c}]?\mathsf{box}.\mathsf{wait}\,x.\mathsf{wait}\,y.\mathsf{done},\\ &\qquad \mathsf{no}.\mathit{Buyer}_1\langle x, y\rangle\},\\ &\mathsf{giveup}.\mathsf{wait}\,y.\lceil x \rceil x[\mathsf{s}]!\mathsf{cancel}.\mathsf{wait}\,x.\mathsf{done}\}\end{aligned}$$

We wish to build a typing derivation showing that Buyer$_1$ *has rank* 1 *and uses* x *and* y *respectively according to* S *and* T*, where* $S = \mathsf{s}!\mathsf{ok}.\mathsf{c}?\mathsf{box}.?\mathsf{end} + \mathsf{s}!\mathsf{cancel}.?\mathsf{end}$ *and* $T = \mathsf{b}_2!\mathsf{split}.(\mathsf{b}_2?\mathsf{yes}.?\mathsf{end} + \mathsf{b}_2?\mathsf{no}.T) + \mathsf{b}_2!\mathsf{giveup}.?\mathsf{end}$*. As it has been noted previously, what makes this process interesting is that it uses the endpoint* x *differently depending on the messages it exchanges with* b_2 *on* y*. Since rule* [TM-TAG] *requires any endpoint other than the one on which messages are exchanged to have the same type, the only way Buyer*$_2$ *can be declared well typed is by means of the casts that occur in its body. For the branch in which Buyer*$_1$ *proposes to* split *the payment we obtain the following derivation tree (we show only the* yes *branch, the* no *one is trivial):*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\emptyset \vdash^0 \mathsf{done}}\,[\textsc{tm-done}]}{y : ?\mathsf{end} \vdash^0 \mathsf{wait}\, y.\mathsf{done}}\,[\textsc{tm-wait}]}{x : ?\mathsf{end}, y : ?\mathsf{end} \vdash^0 \mathsf{wait}\, x \ldots}\,[\textsc{tm-wait}]}{x : \mathsf{c}?\mathsf{box}.?\mathsf{end}, y : ?\mathsf{end} \vdash^0 x[\mathsf{c}]?\mathsf{box} \ldots}\,[\textsc{tm-tag}]}{x : \mathsf{s}!\mathsf{ok}.\mathsf{c}?\mathsf{box}.?\mathsf{end}, y : ?\mathsf{end} \vdash^0 x[\mathsf{s}]!\mathsf{ok} \ldots}\,[\textsc{tm-tag}]}{x : S, y : ?\mathsf{end} \vdash^1 \lceil x \rceil \ldots}\,[\textsc{tm-cast}] \quad \vdots}{x : S, y : \mathsf{b}_2?\mathsf{yes}.?\mathsf{end} + \mathsf{b}_2?\mathsf{no}.T \vdash^1 y[\mathsf{b}_2]?\{\mathsf{yes} \ldots, \mathsf{no} \ldots\}}\,[\textsc{tm-tag}]$$

Note how the application of [TM-CAST] *is key to change the type of* x *in the branch where the proposed split is accepted by* b_2*. In that branch,* x *is deterministically used to send an* ok *message and we leverage on the fair subtyping relation* $S \leqslant_1 \mathsf{s}!\mathsf{ok}.\mathsf{c}?\mathsf{box}.?\mathsf{end}$*.*

For the branch in which Buyer$_1$ *sends* giveup *we obtain the following derivation tree:*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\emptyset \vdash^0 \mathsf{done}}\,[\textsc{tm-done}]}{x : ?\mathsf{end} \vdash^0 \mathsf{wait}\, x.\mathsf{done}}\,[\textsc{tm-wait}]}{x : \mathsf{s}!\mathsf{cancel}.?\mathsf{end} \vdash^0 x[\mathsf{s}]!\mathsf{cancel}.\mathsf{wait}\, x.\mathsf{done}}}{x : S \vdash^1 \lceil x \rceil x[\mathsf{s}]!\mathsf{cancel}.\mathsf{wait}\, x.\mathsf{done}}\,[\textsc{tm-cast}]}{x : S, y : ?\mathsf{end} \vdash^1 \mathsf{wait}\, y.\lceil x \rceil x[\mathsf{s}]!\mathsf{cancel}.\mathsf{wait}\, x.\mathsf{done}}\,[\textsc{tm-wait}]$$

Once again the cast is necessary to change the type of x*, but this time leveraging on the fair subtyping relation* $S \leqslant_1 \mathsf{s}!\mathsf{cancel}.?\mathsf{end}$*. These two derivations can then be combined to complete the proof that the body of*

Buyer$_1$ is well typed:

$$\dfrac{\vdots \qquad\qquad \vdots}{x : S, y : T \vdash^1 y[\mathsf{b}_2]!\{\mathsf{split}\ldots, \mathsf{giveup}\ldots\}}\ [\textsc{tm-tag}]$$

Clearly, it is also necessary to find finite derivation trees for all of the judgments shown above. This can be easily achieved using the corule [COM-TAG]. ⌟

Example 5.2.3. *Casts can be useful to reconcile the types of a channel that is used differently in different branches of a non-deterministic choice. For example, below is an alternative modeling of Buyer from Example 5.1.1 where we abbreviate* seller *to* s *for convenience:*

$$B(x) \triangleq \lceil x\rceil x[\mathsf{s}]!\mathsf{add}.x[\mathsf{s}]!\mathsf{add}.B\langle x\rangle \oplus \lceil x\rceil x[\mathsf{s}]!\mathsf{pay}.\mathsf{close}\, x$$

Note that x is used for sending two add *messages in the left branch of the non-deterministic choice and for sending a single* pay *message in the right branch. Given the session type $S = \mathsf{s}!\mathsf{add}.S + \mathsf{s}!\mathsf{pay}.!\mathsf{end}$ and using the fair subtyping relations $S \leqslant_2 \mathsf{s}!\mathsf{add}.\mathsf{s}!\mathsf{add}.S$ and $S \leqslant_1 \mathsf{s}!\mathsf{pay}.!\mathsf{end}$ we can obtain the following typing derivation for the body of B (we show only the left branch as the right one contains a straightforward application of $S \leqslant_1 \mathsf{s}!\mathsf{pay}.!\mathsf{end}$):*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{x : S \vdash^1 B\langle x\rangle}\ [\textsc{tm-call}]}{x : \mathsf{s}!\mathsf{add}.S \vdash^1 x[\mathsf{s}]!\mathsf{add}.B\langle x\rangle}\ [\textsc{tm-tag}]}{x : \mathsf{s}!\mathsf{add}.\mathsf{s}!\mathsf{add}.S \vdash^1 x[\mathsf{s}]!\mathsf{add}.x[\mathsf{s}]!\mathsf{add}.B\langle x\rangle}\ [\textsc{tm-tag}]}{x : S \vdash^3 \lceil x\rceil x[\mathsf{s}]!\mathsf{add}.x[\mathsf{s}]!\mathsf{add}.B\langle x\rangle}\ [\textsc{tm-cast}] \quad \vdots}{x : S \vdash^1 \lceil x\rceil x[\mathsf{s}]!\mathsf{add}.x[\mathsf{s}]!\mathsf{add}.B\langle x\rangle \oplus \lceil x\rceil x[\mathsf{s}]!\mathsf{pay}.\mathsf{close}\, x}\ [\textsc{tm-choice}]$$

In general, the transformation $u[\mathsf{p}]!\{\mathsf{m}_i.P_i\}_{i=1..n} \rightsquigarrow \lceil u\rceil u[\mathsf{p}]!\mathsf{m}_1.P_1 \oplus \cdots \oplus \lceil u\rceil u[\mathsf{p}]!\mathsf{m}_n.P_n$ does not always preserve typing, so it is not always possible to encode the output of tags using casts and non-deterministic choices. As an example, the definition

$$\mathit{Slot}(x) \triangleq x[\mathsf{p}]?\{\mathsf{play}.x[\mathsf{p}]!\{\mathsf{win}.\mathit{Slot}\langle x\rangle, \mathsf{lose}.\mathit{Slot}\langle x\rangle\}, \mathsf{quit}.\mathsf{close}\, x\}$$

implements the unbiased slot machine of Example 3.2.2 that waits for a message indicating whether a p *wants to* play *another game or to* quit *(we assign role* p *to the player). In the former case, the slot machine notifies* p *of the outcome (either* win *or* lose*). It is easy to see that Slot is well typed under the global type assignment* $\mathit{Slot} : [T; 0]$ *where* $T = \mathsf{p}?\mathsf{play}.(\mathsf{p}!\mathsf{win}.T + \mathsf{p}!\mathsf{lose}.T) + \mathsf{p}?\mathsf{quit}.!\mathsf{end}$*. In particular, Slot has rank 0 since it performs no casts and it creates no sessions. If we encode the tag output in Slot using casts and non-deterministic choices we end up with the following process definition, which is ill typed because it cannot be given a finite rank:*

$$\mathit{Slot}(x) \stackrel{\triangle}{=} x[\mathsf{p}]?\{\mathsf{play}.(\lceil x\rceil x[\mathsf{p}]!\mathsf{win}.\mathit{Slot}\langle x\rangle \oplus \lceil x\rceil x[\mathsf{p}]!\mathsf{lose}.\mathit{Slot}\langle x\rangle), \mathsf{quit}.\mathsf{close}\, x\}$$

The difference between this version of Slot and the above definition of B is that Slot always recurs after a cast, so it is not obvious that finitely many casts suffice in order for Slot to terminate. ⌟

Example 5.2.4. *Example 5.2.2 shows that casts are essential in the type derivation. However, the process would be well typed if we considered a subtyping relation that does not preserve coherence [Gay and Hole, 2005] for the involved types are finite. Now we refine the buyer from Example 5.1.1 in order to consider more involved sessions. Again, we collapse role names to their initials.*

$$\begin{aligned}
B(x) &\stackrel{\triangle}{=} \lceil x\rceil B_1\langle x\rangle \oplus \lceil x\rceil B_2\langle x\rangle\\
B_1(x) &\stackrel{\triangle}{=} x[\mathsf{s}]!\mathsf{add}.x[\mathsf{s}]!\{\mathsf{add}.B_1\langle x\rangle,\ \mathsf{pay}.\mathsf{wait}\, x.\mathsf{done}\}\\
B_2(x) &\stackrel{\triangle}{=} x[\mathsf{s}]!\{\mathsf{add}.x[\mathsf{s}]!\mathsf{add}.B_2\langle x\rangle,\ \mathsf{pay}.\mathsf{wait}\, x.\mathsf{done}\}
\end{aligned}$$

B_2 corresponds to the buyer in Example 5.1.1 while B_1 is the acquirer that adds an odd number of items to the cart. B non deterministically chooses to behave according to B_1 or B_2. Let S_{b_1} and S_{b_2} be the session types such that $x : S_{b_1}$ in B_1 and $x : S_{b_2}$ in B_2 respectively:

$$S_{b_1} = \mathsf{s}!\mathsf{add}.(\mathsf{s}!\mathsf{add}.S_{b_1} + \mathsf{s}!\mathsf{pay}.!\mathsf{end}) \quad S_{b_2} = \mathsf{s}!\mathsf{add}.\mathsf{s}!\mathsf{add}.S_{b_2} + \mathsf{s}!\mathsf{pay}.!\mathsf{end}$$

In example 3.2.1 we showed that $S \leqslant_1 S_{b_2}$ where $S = \mathsf{s}!\mathsf{add}.S + \mathsf{s}!\mathsf{pay}.!\mathsf{end}$ models the acquirer that adds arbitrarily many items to the cart. Analogously, we can prove that $S \leqslant_2 S_{b_1}$. Hence we derive

$$\dfrac{\dfrac{\dfrac{\vdots}{x : S_{b_1} \vdash^0 B_1\langle x\rangle}\,[\text{TM-CALL}]}{x : S \vdash^2 \lceil x\rceil B_1\langle x\rangle}\,[\text{TM-CAST}] \qquad \dfrac{\dfrac{\vdots}{x : S_{b_2} \vdash^0 B_2\langle x\rangle}\,[\text{TM-CALL}]}{x : S \vdash^1 \lceil x\rceil B_2\langle x\rangle}\,[\text{TM-CAST}]}{x : S \vdash^1 \lceil x\rceil B_1\langle x\rangle \oplus \lceil x\rceil B_2\langle x\rangle}\,[\text{TM-CHOICE}]$$

Again, the casts are crucial to obtain the type derivation of process B because rule [TM-CHOICE] *requires that B_1 and B_2 are typed in the same context. Note that B_1 and B_2 are typed with rank 0 since no sessions are created and no casts are performed by the processes.*

Example 5.2.5. *Here we provide evidence that the process definitions in Example 5.1.3 are well typed, even if they model processes that can open arbitrarily many sessions. In that example, the most interesting process definition is that of the worker Sort, which is recursive and may create a new session. In contrast, Merge is finite and Main only refers to Sort. We claim that these process definitions are well typed under the global type assignments*

$$\mathit{Main} : [(); 1] \qquad \mathit{Sort} : [U; 0] \qquad \mathit{Merge} : [T, V; 0]$$

where

$$T = \mathsf{m}!\mathsf{res}.!\mathsf{end} \quad U = \mathsf{m}?\mathsf{req}.T \quad V = \mathsf{w}_1!\mathsf{req}.\mathsf{w}_2!\mathsf{req}.\mathsf{w}_1?\mathsf{res}.\mathsf{w}_2?\mathsf{res}.?\mathsf{end}$$

For the branch of Sort that creates a new session we obtain the derivation tree

$$\dfrac{\dfrac{\vdots}{x : T, t[\mathsf{m}] : V \vdash^0 \mathit{Merge}\langle x, t[\mathsf{m}]\rangle}\,[\text{TM-CALL}] \qquad \dfrac{\vdots}{t[\mathsf{w}_i] : U \vdash^0 \mathit{Sort}\langle t[\mathsf{w}_i]\rangle}}{x : T \vdash^1 (t)(\mathit{Merge}\langle x, t[\mathsf{m}]\rangle \mid \mathit{Sort}\langle t[\mathsf{w}_1]\rangle \mid \mathit{Sort}\langle t[\mathsf{w}_2]\rangle)}\,[\text{TM-PAR}]$$

where $i = 1, 2$. The rank 1 derives from the fact that the created session involves three zero-ranked participants. For the body of Sort we obtain the following derivation tree:

$$\dfrac{\dfrac{\dfrac{\vdots}{x : T \vdash^1 (t)(\mathit{Merge}\langle x, t[\mathsf{m}]\rangle \mid \cdots)}\,[\text{TM-PAR}] \qquad \dfrac{\dfrac{}{x : !\mathsf{end} \vdash^0 \mathsf{close}\, x}\,[\text{TM-CLOSE}]}{x : T \vdash^0 x[\mathsf{m}]!\mathsf{res}\ldots}\,[\text{TM-TAG}]}{x : T \vdash^0 (t)(\mathit{Merge}\langle x, t[\mathsf{m}]\rangle \mid \cdots) \oplus x[\mathsf{m}]!\mathsf{res}\ldots}\,[\text{TM-CHOICE}]}{x : U \vdash^0 x[\mathsf{m}]?\mathsf{req}.((t)(\mathit{Merge}\langle x, t[\mathsf{m}]\rangle \mid \cdots) \oplus x[\mathsf{m}]!\mathsf{res}\ldots)}\,[\text{TM-TAG}]$$

In the application of the rule [TM-CHOICE]*, the rank of the whole choice coincides with that of the branch in which no new sessions are created. This way we account for the fact that, even though Sort* may *create a new session, it does not* have to *do so in order to terminate.* ⌟

5.3 Correctness

In this section we discuss the proof of Theorem 5.2.1. The proof technique follows that of Theorem 4.2.1. Such proof is essentially composed of a standard subject reduction result showing that typing is preserved by reductions and a proof that every well-typed process other than done may always reduce in such a way that a suitably defined *well-founded measure* strictly decreases. The measure is a lexicographically ordered pair of natural numbers with the following meaning: the first component measures the number of sessions that must be created and the total weight of casts that must be performed in order for the process to terminate (this information is essentially the rank we associate with typing judgments); the second component measures the overall effort required to terminate every session that has already been created (these sessions are identified by the fact that their restriction occurs unguarded in the process). We account for this effort by measuring the shortest reduction that terminates a coherent session map (Definition 2.3.2). The reason why we need two quantities in the measure is that in general every application of fair subtyping may *increase* the length of the shortest reduction that terminates a coherent session map. So, when casts are performed the second component of the measure may increase, but the first component reduces. As a final remark, it should be noted that the overall measure associated with a well-typed process *may also increase*, for example if new sessions are created (Example 5.1.3). However, one particular reduction that decreases the measure is always guaranteed to exist. For the sake of clarity, in the following we often write M coherent instead of $\#M$.

5.3.1 Subject Reduction

We show the proof of a standard subject reduction theorem (see Lemma 5.3.3). For this sake, we need an additional result, that we dub subject congruence, stating that well-typedness is preserved by the structural precongruence relation for processes in Figure 5.2 (Lemma 5.3.2). Notably, Lemma 5.3.2 tells that the rank does not incrase. Such result is not provable in Lemma 5.3.3 as the non deterministic choice can reduce to a branch that increases the rank.

Lemma 5.3.1. If $\Gamma, x : S \vdash^n P$ and $\Gamma, u : S$ is defined, then $\Gamma, u : S \vdash^n P\{u/x\}$. A typing context is *defined* if the endpoints occurring in it all have different session names.

Proof. By bounded coinduction (see Proposition 1.2.3). □

Lemma 5.3.2 (Subject Congruence)**.** If $\Gamma \vdash^n P$ and $P \preccurlyeq Q$, then $\Gamma \vdash^m Q$ for some $m \leq n$.

Proof. By induction on the derivation of $P \preccurlyeq Q$ and by cases on the last rule applied.

Case [SM-PAR-COMM]. Then $P = (s)(\overline{P}|P'|Q'|\overline{Q}) \preccurlyeq (s)(\overline{P}|Q'|P'|\overline{Q}) = Q$. From rule [TM-PAR] we deduce that there exist $\Gamma_i, \mathsf{p}_i, S_i, n_i$ for $i = 1, \dots, h$ such that

- $\Gamma = \Gamma_1, \dots, \Gamma_h$
- $n = 1 + \sum_{i=1}^{h} n_i$
- $\prod_{i=1}^{h} \mathsf{p}_i \triangleright S_i$ coherent
- $\Gamma_i, s[\mathsf{p}_i] : S_i \vdash^{n_i} P_i$ for $i = 1, \dots, k$
- $\Gamma_{k+1}, s[\mathsf{p}_{k+1}] : S_{k+1} \vdash^{n_{k+1}} P'$
- $\Gamma_{k+2}, s[\mathsf{p}_{k+2}] : S_{k+2} \vdash^{n_{k+2}} Q'$
- $\Gamma_i, s[\mathsf{p}_i] : S_i \vdash^{n_i} Q_i$ for $i = k+3, \dots, h$

We conclude $\Gamma \vdash^m Q$ with one application of [TM-PAR] by taking $m \overset{\mathsf{def}}{=} n$.

Case [SM-PAR-ASSOC]. Then $P = (s)(\overline{P}|(t)(R|\overline{Q})) \preccurlyeq (t)((s)(\overline{P}|R)|\overline{Q}) = Q$ and $s \in \mathsf{fn}(R)$. From rule [TM-PAR] we deduce that there exist $\Gamma_i, \mathsf{p}_i, S_i, n_i$ for $i = 1, \dots, h$ such that

- $\Gamma = \Gamma_1, \dots, \Gamma_h$
- $n = 1 + \sum_{i=1}^{h} n_i$
- $\prod_{i=1}^{h} \mathsf{p}_i \triangleright S_i$ coherent
- $\Gamma_i, s[\mathsf{p}_i] : S_i \vdash^{n_i} P_i$ for $i = 1, \dots, h-1$
- $\Gamma_h, s[\mathsf{p}_h] : S_h \vdash^{n_h} (t)(R \mid \overline{Q})$

From rule [TM-PAR] and the hypothesis that $s \in \mathsf{fn}(R)$ we deduce that there exist $\Delta_i, \mathsf{q}_i, T_i, m_i$ for $i = 1, \dots, k$ such that

- $\Gamma_h = \Delta_1, \dots, \Delta_k$
- $n_h = 1 + \sum_{1}^{k} m_i$

- $\prod_1^k \mathsf{q}_i \triangleright T_i$ coherent
- $\Delta_1, s[\mathsf{p}_h] : S_h, t[\mathsf{q}_1] : T_1 \vdash^{m_1} R$
- $\Delta_{i+1}, t[\mathsf{q}_{i+1}] : T_{i+1} \vdash^{m_{i+1}} Q_i$ for $i = 1, \dots, k-1$

Using [TM-PAR] we deduce $\Gamma_1, \dots, \Gamma_{h-1}, \Delta_1, t[\mathsf{q}_1] : T_1 \vdash^{1+\sum_{i=1}^{h-1} n_i + m_1} (s)(\overline{P} \mid R)$. We conclude $\Gamma \vdash^m (t)((s)(\overline{P}|R)|\overline{Q})$ with another application of [TM-PAR] by taking $m \overset{\text{def}}{=} n$.

Case [SM-CAST-COMM]. Then $P = \lceil u \rceil \lceil v \rceil R \preccurlyeq \lceil v \rceil \lceil u \rceil R = Q$. We can assume $u \neq v$ or else $P = Q$. From rule [TM-CAST] we deduce that there exist Γ_1, S, T, n_1, n_u such that

- $\Gamma = \Gamma_1, u : S$
- $S \leqslant_{n_u} T$
- $n = n_u + n_1$
- $\Gamma_1, u : T \vdash^{n_1} \lceil v \rceil R$

From rule [TM-CAST] we deduce that there exist $\Gamma_2, S', T', n_2, n_v$ such that

- $\Gamma_1 = \Gamma_2, v : S'$
- $S' \leqslant_{n_v} T'$
- $n_1 = n_v + n_2$
- $\Gamma_2, u : T, v : T' \vdash^{n_2} R$

We derive $\Gamma_2, u : S, v : T' \vdash^{n_u + n_2} \lceil u \rceil R$ with one application of [TM-CAST] and we conclude with another application of [TM-CAST] by taking $m \overset{\text{def}}{=} n$.

Case [SM-CAST-NEW]. Then $P = (s)(\lceil s[\mathsf{p}] \rceil R \mid \overline{P}) \preccurlyeq (s)(R \mid \overline{P}) = Q$. From rule [TM-PAR] we deduce that there exist Δ, n' and $\Gamma_i, \mathsf{q}_i, n_i$ for $i = 1, \dots, h$ such that

- $\Gamma = \Delta, \Gamma_1, \dots, \Gamma_h$ for $i = 1, \dots, h$
- $n = 1 + n' + \sum_{i=1}^h n_i$
- $\mathsf{p} \triangleright S \mid \prod_{i=1}^h \mathsf{q}_i \triangleright S_i$ coherent
- $\Delta, s[\mathsf{p}] : S \vdash^{n'} \lceil s[\mathsf{p}] \rceil R$
- $\Gamma_i, s[\mathsf{q}_i] : S_i \vdash^{n_i} P_i$ for $i = 1, \dots, h$

From rule [TM-CAST] we deduce that there exist T, m', m_s such that

- $S \leqslant_{m_s} T$
- $n' = m_s + m'$
- $\Delta, s[\mathsf{p}] : T \vdash^{m'} R$

From $\mathsf{p} \triangleright S \mid \prod_{i=1}^{h} \mathsf{q}_i \triangleright S_i$ coherent, $S \leqslant_{m_s} T$ and definition 3.2.1 we deduce $\mathsf{p} \triangleright T \mid \prod_{i=1}^{h} \mathsf{q}_i \triangleright S_i$ coherent. We conclude with an application of [TM-PAR] by taking $m \overset{\mathsf{def}}{=} 1 + m' + \sum_{i=1}^{h} n_i \leq n$.

Case [SM-CAST-SWAP]. Then $P = (s)(\lceil t[\mathsf{p}] \rceil R \mid \overline{P}) \preccurlyeq \lceil t[\mathsf{p}] \rceil (s)(R \mid \overline{P}) = Q$ and $t \neq s$. From rule [TM-PAR] we deduce that there exist $\Gamma_i, \mathsf{q}_i, n_i$ for $i = 1, \dots, h$ such that

- $\Gamma = \Gamma_1, \dots, \Gamma_h$ for $i = 1, \dots, h$
- $n = 1 + \sum_{i=1}^{h} n_i$
- $\prod_{i=1}^{h} \mathsf{q}_i \triangleright S_i$ coherent
- $\Gamma_1, s[\mathsf{q}_1] : S_1 \vdash^{n_1} \lceil t[\mathsf{p}] \rceil R$
- $\Gamma_i, s[\mathsf{q}_i] : S_i \vdash^{n_i} P_i$ for $i = 2, \dots, h$

From rule [TM-CAST] we deduce that there exist Δ, T, n', m_t such that

- $\Gamma_1 = \Delta, t[\mathsf{p}] : S$
- $S \leqslant_{m_t} T$
- $n_1 = m_t + n'$
- $\Delta, t[\mathsf{p}] : T, s[\mathsf{q}_1] : S_1 \vdash^{n'} R$

We derive $\Delta, t[\mathsf{p}] : T, \Gamma_2, \dots, \Gamma_h \vdash^{1+n'+\sum_{i=2}^{h} n_i} (s)(R \mid \overline{P})$ with an application of [TM-PAR]. We conclude with an application of [TM-CAST] by taking $m \overset{\mathsf{def}}{=} n$.

Case [SM-CALL]. Then $P = A\langle \overline{u} \rangle \preccurlyeq R\{\overline{u}/\overline{x}\} = Q$ and $A(\overline{x}) \overset{\triangle}{=} R$. From [TM-CALL] we conclude that there exist $\overline{S}$ and m such that $A : [\overline{S}; m]$ and $\Gamma = \overline{u : S}$ and $\overline{u : S} \vdash^{m} Q$ and $m \leq n$. □

Lemma 5.3.3 (Subject Reduction)**.** If $\Gamma \vdash^{n} P$ and $P \to Q$, then $\Gamma \vdash^{m} Q$ for some m.

Proof. By induction on the derivation of $P \to Q$ and by cases on the last rule applied.

Case [RM-CHOICE]. Then $P = P_1 \oplus P_2 \to P_k = Q$ and $k \in \{1, 2\}$. From [TM-CHOICE] we deduce that $\Gamma \vdash^m Q$ for some m.

Case [RM-SIGNAL]. Then $P = (s)(\mathsf{wait}\, s[\mathsf{p}].Q | \mathsf{close}\, s[\mathsf{q}_1] | \cdots | \mathsf{close}\, s[\mathsf{q}_h]) \to Q$. From [TM-PAR], [TM-WAIT] and [TM-CLOSE] we deduce that there exist m and n_i for $i = 1, \dots, h$ such that

- $n = 1 + m + \sum_{i=1}^{h} n_i$
- $\Gamma, s[\mathsf{p}] : ?\mathsf{end} \vdash^m \mathsf{wait}\, s[\mathsf{p}].Q$
- $\Gamma \vdash^m Q$
- $s[\mathsf{q}_i] : !\mathsf{end} \vdash^{n_i} \mathsf{close}\, s[\mathsf{q}_i]$ for $i = 1, \dots, h$

There is nothing left to prove.

Case [RM-CHANNEL]. Then $P = (s)(s[\mathsf{p}][\mathsf{q}]!v.P' \mid s[\mathsf{q}][\mathsf{p}]?(x).Q' \mid \overline{R}) \to (s)(P' \mid Q'\{v/x\} \mid \overline{R}) = Q$. From [TM-PAR] we deduce that there exist $\Gamma_i, S_i, \mathsf{p}_i, n_i$ for $i = 1, \dots, h$ such that

- $\Gamma = \Gamma_1, \dots, \Gamma_h$
- $n = 1 + \sum_{i=1}^{h} n_i$
- $\prod_{i=1}^{h} \mathsf{p}_i \triangleright S_i$ coherent
- $\mathsf{p} = \mathsf{p}_1$ and $\mathsf{q} = \mathsf{p}_2$
- $\Gamma_1, s[\mathsf{p}] : S_1 \vdash^{n_1} s[\mathsf{p}][\mathsf{q}]!v.P'$
- $\Gamma_2, s[\mathsf{q}] : S_2 \vdash^{n_2} s[\mathsf{q}][\mathsf{p}]?(x).Q'$
- $\Gamma_i, s[\mathsf{p}_i] : S_i \vdash^{n_i} R_i$ for $i = 3, \dots, h$

From [TM-CHANNEL-OUT] and [TM-CHANNEL-IN] we deduce that there exist S_v, T_1, T_2, Δ_1 such that

- $S_1 = \mathsf{q}!S_v.T_1$
- $\Gamma_1 = \Delta_1, v : S_v$
- $\Delta_1, s[\mathsf{p}] : T_1 \vdash^{n_1} P'$
- $S_2 = \mathsf{p}?S_v.T_2$

- $\Gamma_2, s[\mathsf{q}] : T_2, x : S_v \vdash^{n_2} Q'$

Using lemma 5.3.1 we deduce $\Gamma_2, s[\mathsf{q}] : T_2, v : S_v \vdash^{n_2} Q'\{v/x\}$. Using definition 2.3.2 we deduce $\mathsf{p} \triangleright T_1 \mid \mathsf{q} \triangleright T_2 \mid \prod_{i=3}^{h} \mathsf{p}_i \triangleright S_i$ coherent. We conclude with one application of [TM-PAR] taking $m \overset{\mathsf{def}}{=} n$.

Case [RM-PICK]. Then $P = (s)(s[\mathsf{p}][\mathsf{q}]!\{\mathsf{m}_i.P_i\}_{i \in I}|\overline{Q}) \to (s)(s[\mathsf{p}][\mathsf{q}]!\mathsf{m}_k.P_k| \overline{Q}) = Q$ and $k \in I$. From [TM-PAR] we deduce that there exist $\Gamma_i, \mathsf{p}_i, n_i, S_i$ for $i = 1, \ldots, h$ such that

- $\Gamma = \Gamma_1, \ldots, \Gamma_h$
- $n = 1 + \sum_{i=1}^{h} n_i$
- $\prod_{i=1}^{h} \mathsf{p}_i \triangleright S_i$ coherent
- $\mathsf{p} = \mathsf{p}_1$ and $\mathsf{q} = \mathsf{p}_i$ for some $i \in \{2, \ldots, h\}$
- $\Gamma_1, s[\mathsf{p}] : S_1 \vdash^{n_1} s[\mathsf{p}][\mathsf{q}]!\{\mathsf{m}_i.P_i\}_{i \in I}$
- $\Gamma_i, s[\mathsf{p}_i] : S_i \vdash^{n_i} Q_i$ for $i = 2, \ldots, h$

From [TM-TAG] we deduce that there exist T_i for all $i \in I$ such that

- $S_1 = \sum_{i \in I} \mathsf{q}!\mathsf{m}_i.T_i$
- $\Gamma_1, s[\mathsf{p}] : T_i \vdash^{n_1} P_i \ ^{(i \in I)}$

From the hypothesis that $k \in I$ we deduce that $\Gamma_1, s[\mathsf{p}] : T_k \vdash^{n_1} P_k$ and from [TM-TAG] we deduce $\Gamma_1, s[\mathsf{p}] : \mathsf{q}!\mathsf{m}_k.T_k \vdash^{n_1} s[\mathsf{p}][\mathsf{q}]!\mathsf{m}_k.P_k$. From definition 2.3.2 we deduce that $\mathsf{q}!\mathsf{m}_k.T_k \mid \prod_{i=2}^{h} \mathsf{p}_i \triangleright S_i$ coherent. We conclude with an application of [TM-PAR] taking $m \overset{\mathsf{def}}{=} n$.

Case [RM-TAG]. Then $P = (s)(s[\mathsf{p}][\mathsf{q}]!\mathsf{m}_k.P' \mid s[\mathsf{q}][\mathsf{p}]?\{\mathsf{m}_i.Q_i\}_{i \in I} \mid \overline{R}) \to (s)(P' \mid Q_k \mid \overline{R}) = Q$ and $k \in I$. From [TM-PAR] we deduce that there exist $\Gamma_i, S_i, \mathsf{p}_i, n_i$ for $i = 1, \ldots, h$ such that

- $\Gamma = \Gamma_1, \ldots, \Gamma_h$
- $n = 1 + \sum_{i=1}^{h} n_i$
- $\prod_{i=1}^{h} \mathsf{p}_i \triangleright S_i$ coherent
- $\mathsf{p} = \mathsf{p}_1$ and $\mathsf{q} = \mathsf{p}_2$
- $\Gamma_1, s[\mathsf{p}] : S_1 \vdash^{n_1} s[\mathsf{p}][\mathsf{q}]!\mathsf{m}_k.P'$
- $\Gamma_2, s[\mathsf{q}] : S_2 \vdash^{n_2} s[\mathsf{q}][\mathsf{p}]?\{\mathsf{m}_i.Q_i\}_{i \in I}$

- $\Gamma_i, s[\mathsf{p}_i] : S_i \vdash^{n_i} R_i$ for $i = 3, \dots, h$

From [TM-TAG] we deduce that there exist S_1' and T_i for every $i \in I$ such that

- $S_1 = \mathsf{q}!\mathsf{m}_k.S_1'$
- $\Gamma_1, s[\mathsf{p}] : S_1' \vdash^{n_1} P'$
- $S_2 = \sum_{i\in I} \mathsf{p}?\mathsf{m}_i.T_i$
- $\Gamma_2, s[\mathsf{q}] : T_i \vdash^{n_2} Q_i \ ^{(i\in I)}$

From definition 2.3.2 we deduce that $\mathsf{p} \triangleright S_1' \mid \mathsf{q} \triangleright T_k \mid \prod_{i=2}^{h} \mathsf{p}_i \triangleright S_i$ coherent. We conclude with an application of [TM-PAR] by taking $m \overset{\mathsf{def}}{=} n$.

Case [RM-PAR]. Then $P = (s)(P' \mid \overline{R}) \to (s)(Q' \mid \overline{R}) = Q$ and $P' \to Q'$. From [TM-PAR] we deduce that there exist $\Gamma_i, \mathsf{p}_i, S_i, n_i$ for $i = 1, \dots, h$ such that

- $\Gamma = \Gamma_1, \dots, \Gamma_h$
- $n = 1 + \sum_{i=1}^{h} n_i$
- $\prod_{i=1}^{h} \mathsf{p}_i \triangleright S_i$ coherent
- $\Gamma_1, s[\mathsf{p}_1] : S_1 \vdash^{n_1} P'$
- $\Gamma_i, s[\mathsf{p}_i] : S_i \vdash^{n_i} R_i$ for $i = 2, \dots, h$

Using the induction hypothesis on $\Gamma_1, s[\mathsf{p}_1] : S_1 \vdash^{n_1} P'$ and $P' \to Q'$ we deduce $\Gamma_1, s[\mathsf{p}_1] : S_1 \vdash^{n_1'} Q'$ for some n_1'. We conclude with an application of [TM-PAR] taking $m \overset{\mathsf{def}}{=} 1 + n_1' + \sum_{i=2}^{h} n_i$.

Case [RM-CAST]. Then $P = \lceil u \rceil P' \to \lceil u \rceil Q' = Q$ and $P' \to Q'$. From [TM-CAST] we deduce that there exist S, T, Γ', n', m_u such that

- $\Gamma = \Gamma', u : S$
- $S \leqslant_{m_u} T$
- $n = m_u + n'$
- $\Gamma', u : T \vdash^{n'} P'$

MTM-THREAD

$$\frac{}{\Gamma \vDash^{(n,0)} P} \quad \Gamma \vdash^{n} P$$

MTM-CAST

$$\frac{\Gamma, u : T \vDash^{\mu} P}{\Gamma, u : S \vDash^{\mu+(n,0)} \lceil u \rceil P} \; S \leqslant_n T$$

MTM-PAR

$$\frac{\Gamma_i, s[\mathsf{p}_i] : S_i \vDash^{\mu_i} P_i \;\; {}^{(i=1,\dots,h)}}{\Gamma_1, \dots, \Gamma_h \vDash^{\sum_{i=1}^{h} \mu_i + (0, \|\{\mathsf{p}_i \triangleright S_i\}_{i=1,\dots,h}\|)} (s)(P_1 \mid \cdots \mid P_h)} \; \#\{\mathsf{p}_i \triangleright S_i\}_{i=1..h}$$

Figure 5.5: Typing rules with measure

Using the induction hypothesis on $\Gamma', u : T \vdash^{n'} P'$ and $P' \to Q'$ we deduce $\Gamma', u : T \vdash^{m'} Q'$ for some m'. We conclude with an application of [TM-CAST] taking $m \overset{\mathsf{def}}{=} m_u + m'$.

Case [RM-STRUCT]. Then $P \preccurlyeq P' \to Q' \preccurlyeq Q$. From Lemma 5.3.2 we deduce that $\Gamma \vdash^{n'} P'$ for some $n' \leq n$. Using the induction hypothesis on $\Gamma \vdash^{n'} P'$ and $P' \to Q'$ we deduce $\Gamma \vdash^{m'} Q'$ for some m'. We conclude using lemma 5.3.2 once more. $\square$

5.3.2 Measure

𝄢 We introduce two fundamental notions for the soundness proof of the type system. First, we introduce the *rank* of a session map M as the minimum length to reach successful termination of session M. Then, we introduce the *measure* of a process which takes into account the rank in the typing judgment as well as the ranks of the session that have been already opened. We embed such measure in the typing derivations by using a refined set of rules. At last, we compare the typing judgments labeled with the usual rank with those including the measure (see Lemma 5.3.4) and we prove that structural precongruence of processes does not increase the measure (see Lemma 5.3.5).

Definition 5.3.1 (rank)**.** The *rank* of a session map $M = \prod_{i=1}^{h} \mathsf{p}_i \triangleright S_i$, written $\|M\|$, is the element of $\mathbb{N} \cup \{\infty\}$ defined as

$$\|M\| \overset{\mathsf{def}}{=} \min |M \overset{?\checkmark}{\Longrightarrow}|$$

where $|M \overset{\alpha}{\Longrightarrow} N|$ denotes the length of the sequence $\tau, \dots, \tau, \alpha$ and we postulate that $\min \emptyset = \infty$.

Definition 5.3.2 (Measure)**.** The measure of a process is a lexicographically ordered pair of natural numbers (m, n) where:

- m is an upper bound to the number of sessions that the process may open and of weights of casts that the process may perform *in the future* before it terminates;
- n is the overall effort for terminating the sessions that have been already opened *in the past*, *i.e.* the sum of their rank (Definition 5.3.1).

In Figure 5.5 we introduce a refined set of typing rules for processes that allow us to associate them with their measure, not just with their rank. The idea behind these rules (similarly to Figure 4.5) is that they distinguish between *past* and *future* of a process by looking at its structure. Indeed, unguarded sessions have been created, casts have not been performed yet and sessions that occur guarded have not been created yet. [MTM-THREAD] adopts the rank of the process inside the usual typing judgment (Figure 5.4) as first component of the measure. This rule has lower priority with respect to the other rules so that it is applied to processes that are not casts or restrictions. In [MTM-CAST] the first component of the measure is increased by the weight of the cast. [MTM-PAR] increases the second component of the measure by the rank of the involved session.

Lemma 5.3.4. The following properties hold:

1. $\Gamma \vdash^n P$ implies $\Gamma \vDash^\mu P$ for some $\mu \leq (n, 0)$;
2. $\Gamma \vDash^\mu P$ implies $\Gamma \vdash^n P$ for some n such that $\mu \leq (n, 0)$.

Proof. We prove item 1 by induction on the structure of P. The proof of item 2 is by a straightforward induction over $\Gamma \vDash^\mu P$.

Case $P = (s)(\overline{P})$. From [TM-PAR] we deduce that there exist $\Gamma_i, \mathsf{p}_i, S_i, n_i$ for $i = 1, \dots, h$ such that

- $\Gamma = \Gamma_1, \dots, \Gamma_h$
- $n = 1 + \sum_{i=1}^h n_i$
- $\prod_{i=1}^h \mathsf{p}_i \triangleright S_i$ coherent
- $\Gamma_i, s[\mathsf{p}_i] : S_i \vdash^{n_i} P_i \ ^{(i=1,\dots,h)}$

Using the induction hypothesis on $\Gamma_i, s[\mathsf{p}_i] : S_i \vdash^{n_i} P_i \ ^{(i=1,\dots,h)}$ we deduce that there exist μ_i for $i = 1, \dots, h$ such that

- $\Gamma_i, s[\mathsf{p}_i] : S_i \vDash^{\mu_i} P_i \ ^{(i=1,\dots,h)}$
- $\mu_i \leq (n_i, 0)$ for $i = 1, \dots, h$

We conclude with one application of [MTM-PAR] by taking $\mu \overset{\text{def}}{=} \sum_{i=1}^{h} \mu_i + (0, \| \prod_{i=1}^{h} \mathsf{p}_i \triangleright S_i \|)$ and observing that $\mu < (n_1, 0) + (n_2, 0) + \cdots + (n_h, 0) + (1, 0) = (n, 0)$.

Case $P = \lceil u \rceil Q$. From [TM-CAST] we deduce that there exist Δ, S, T, m and m_u such that

- $\Gamma = \Delta, u : S$
- $S \leqslant_{m_u} T$
- $n = m_u + m$
- $\Delta, u : T \vdash^m Q$

Using the induction hypothesis on $\Delta, u : T \vdash^m Q$ we deduce $\Delta, u : T \vDash^\nu Q$ for some $\nu \leq (m, 0)$. We conclude with an application of [MTM-CAST] by taking $\mu \overset{\text{def}}{=} \nu + (m_u, 0)$ and observing that $\mu \leq (m, 0) + (m_u, 0) = (n, 0)$.

In all the other cases. We conclude with an application of [MTM-THREAD] by taking $\mu \overset{\text{def}}{=} (n, 0)$. □

Lemma 5.3.5. If $\Gamma \vDash^\mu P$ and $P \preccurlyeq Q$, then there exists $\nu \leq \mu$ such that $\Gamma \vDash^\nu Q$.

Proof. By induction on the derivation of $P \preccurlyeq Q$ and by cases on the last rule applied. We only consider the base cases.

Case [SM-PAR-COMM]. Then $P = (s)(\overline{P} | P' | Q' | \overline{Q}) \preccurlyeq (s)(\overline{P} | Q' | P' | \overline{Q}) = Q$. From rule [MTM-PAR] we deduce that there exist $\Gamma_i, \mathsf{p}_i, S_i, \mu_i$ for $i = 1, \dots, h$ such that

- $\Gamma = \Gamma_1, \dots, \Gamma_h$
- $\mu = \sum_{i=1}^{h} \mu_i + (0, \| \prod_{i=1}^{h} \mathsf{p}_i \triangleright S_i \|)$
- $\prod_{i=1}^{h} \mathsf{p}_i \triangleright S_i$ coherent
- $\Gamma_i, s[\mathsf{p}_i] : S_i \vDash^{\mu_i} P_i$ for $i = 1, \dots, k$
- $\Gamma_{k+1}, s[\mathsf{p}_{k+1}] : S_{k+1} \vDash^{\mu_{k+1}} P'$
- $\Gamma_{k+2}, s[\mathsf{p}_{k+2}] : S_{k+2} \vDash^{\mu_{k+2}} Q'$
- $\Gamma_i, s[\mathsf{p}_i] : S_i \vDash^{\mu_i} Q_i$ for $i = k + 3, \dots, h$

We conclude $\Gamma \vDash^{\nu} Q$ with one application of [MTM-PAR] by taking $\nu \overset{\text{def}}{=} \mu$.

Case [SM-PAR-ASSOC]. Then $P = (s)(\overline{P}|(t)(R|\overline{Q})) \preccurlyeq (t)((s)(\overline{P}|R)|\overline{Q}) = Q$ and $s \in \mathsf{fn}(R)$. From rule [MTM-PAR] we deduce that there exist $\Gamma_i, \mathsf{p}_i, S_i, \mu_i$ for $i = 1, \dots, h$ such that

- $\Gamma = \Gamma_1, \dots, \Gamma_h$
- $\mu = \sum_{i=1}^{h} \mu_i + (0, \| \prod_{i=1}^{h} \mathsf{p}_i \triangleright S_i \|)$
- $\prod_{i=1}^{h} \mathsf{p}_i \triangleright S_i$ coherent
- $\Gamma_i, s[\mathsf{p}_i] : S_i \vDash^{\mu_i} P_i$ for $i = 1, \dots, h-1$
- $\Gamma_h, s[\mathsf{p}_h] : S_h \vDash^{\mu_h} (t)(R \mid \overline{Q})$

From rule [MTM-PAR] and the hypothesis that $s \in \mathsf{fn}(R)$ we deduce that there exist $\Delta_i, \mathsf{q}_i, T_i, \nu_i$ for $i = 1, \dots, k$ such that

- $\Gamma_h = \Delta_1, \dots, \Delta_k$
- $\mu_h = \sum_1^k \nu_i + (0, \| \prod_{i=1}^{k} \mathsf{q}_i \triangleright T_i \|)$
- $\prod_{i=1}^{k} \mathsf{q}_i \triangleright T_i$ coherent
- $\Delta_1, s[\mathsf{p}_h] : S_h, t[\mathsf{q}_1] : T_1 \vDash^{\nu_1} R$
- $\Delta_{i+1}, t[\mathsf{q}_{i+1}] : T_{i+1} \vDash^{\nu_{i+1}} Q_i$ for $i = 1, \dots, k-1$

Using [TM-PAR] we deduce

- $\Gamma_1, \dots, \Gamma_{h-1}, \Delta_1, t[\mathsf{q}_1] : T_1 \vDash^{\sum_{i=1}^{h-1} \mu_i + \nu_1 + \| \prod_{i=1}^{h} \mathsf{p}_i \triangleright S_i \|} (s)(\overline{P} \mid R)$

We conclude $\Gamma \vDash^{\nu} (t)((s)(\overline{P} \mid R) \mid \overline{Q})$ with another application of [MTM-PAR] by taking $\nu \overset{\text{def}}{=} \mu$.

Case [SM-CAST-COMM]. Then $P = \lceil u \rceil \lceil v \rceil R \preccurlyeq \lceil v \rceil \lceil u \rceil R = Q$. We can assume $u \neq v$ or else $P = Q$. From rule [MTM-CAST] we deduce that there exist $\Gamma_1, S, T, \mu_1, m_u$ such that

- $\Gamma = \Gamma_1, u : S$
- $S \leqslant_{m_u} T$
- $\mu = \mu_1 + (m_u, 0)$
- $\Gamma_1, u : T \vDash^{\mu_1} \lceil v \rceil R$

From rule [TM-CAST] we deduce that there exist $\Gamma_2, S', T', \mu_2, m_v$ such that

- $\Gamma_1 = \Gamma_2, v : S'$
- $S' \leqslant_{m_v} T'$
- $\mu_1 = \mu_2 + (m_v, 0)$
- $\Gamma_2, u : T, v : T' \vDash^{\mu_2} R$

We derive $\Gamma_2, u : S, v : T' \vDash^{\mu_2 + (m_u, 0)} \lceil u \rceil R$ with one application of [MTM-CAST] and we conclude with another application of [MTM-CAST] by taking $\nu \overset{\text{def}}{=} \mu$.

Case [SM-CAST-NEW]. Then $P = (s)(\lceil s[\mathsf{p}] \rceil R \mid \overline{P}) \preccurlyeq (s)(R \mid \overline{P}) = Q$. From rule [MTM-PAR] we deduce that there exist Δ, μ', S and $\Gamma_i, \mathsf{q}_i, S_i, \mu_i$ for $i = 1, \dots, h$ such that

- $\Gamma = \Delta, \Gamma_1, \dots, \Gamma_h$
- $\mu = \mu' + \sum_{i=1}^{h} \mu_i + (0, \|\mathsf{p} \triangleright S \mid \prod_{i=1}^{h} \mathsf{q}_i \triangleright S_i\|)$
- $\mathsf{p} \triangleright S \mid \prod_{i=1}^{h} \mathsf{q}_i \triangleright S_i$ coherent
- $\Delta, s[\mathsf{p}] : S \vDash^{\mu'} \lceil s[\mathsf{p}] \rceil R$
- $\Gamma_i, s[\mathsf{q}_i] : S_i \vDash^{\mu_i} P_i$ for $i = 1, \dots, h$

From rule [MTM-CAST] we deduce that there exist T, ν', m_s such that

- $S \leqslant_{m_s} T$
- $\nu' = \mu' + (m_s, 0)$
- $\Delta, s[\mathsf{p}] : T \vDash^{\nu'} R$

From $\mathsf{p} \triangleright S \mid \prod_{i=1}^{h} \mathsf{q}_i \triangleright S_i$ coherent, $S \leqslant_{m_s} T$ and definition 3.2.1 we deduce $\mathsf{p} \triangleright T \mid \prod_{i=1}^{h} \mathsf{q}_i \triangleright S_i$ coherent. We conclude with an application of [MTM-PAR] by taking $\nu = \nu' + \sum_{i=1}^{h} \mu_i + (0, \|\mathsf{p} \triangleright S \mid \prod_{i=1}^{h} \mathsf{q}_i \triangleright S_i\|) \leq n$.

Case [SM-CAST-SWAP]. Then $P = (s)(\lceil t[\mathsf{p}] \rceil R \mid \overline{P}) \preccurlyeq \lceil t[\mathsf{p}] \rceil (s)(R \mid \overline{P}) = Q$ and $t \neq s$. From rule [MTM-PAR] we deduce that there exist $\Gamma_i, \mathsf{q}_i, \mu_i$ for $i = 1, \dots, h$ such that

- $\Gamma = \Gamma_1, \dots, \Gamma_h$
- $\mu = \sum_{i=1}^{h} \mu_i + (0, \| \prod_{i=1}^{h} \mathsf{q}_i \triangleright S_i\|)$

- $\prod_{i=1}^{h} \mathsf{q}_i \triangleright S_i$ coherent
- $\Gamma_1, s[\mathsf{q}_1] : S_1 \vDash^{\mu_1} \lceil t[\mathsf{p}] \rceil R$
- $\Gamma_i, s[\mathsf{q}_i] : S_i \vDash^{\mu_i} P_i$ for $i = 2, \dots, h$

From rule [MTM-CAST] we deduce that there exist Δ, T, μ', m_t such that

- $\Gamma_1 = \Delta, t[\mathsf{p}] : S$
- $S \leqslant_{m_t} T$
- $\mu_1 = \mu' + (m_t, 0)$
- $\Delta, t[\mathsf{p}] : T, s[\mathsf{q}_1] : S_1 \vDash^{\mu'} R$

We derive $\Delta, t[\mathsf{p}] : T, \Gamma_2, \dots, \Gamma_h \vDash^{\mu' + \sum_{i=2}^{h} \mu_i + (0, \| \prod_{i=1}^{h} \mathsf{q}_i \triangleright S_i \|)} (s)(R \mid \overline{P})$ with an application of [MTM-PAR]. We conclude with an application of [MTM-CAST] by taking $m \overset{\mathsf{def}}{=} n$.

Case [SM-CALL]. Then $P = A\langle \overline{u} \rangle \preccurlyeq R\{\overline{x}/\overline{u}\} = Q$ and $A(\overline{x}) \overset{\triangle}{=} R$. From [MTM-THREAD] we deduce that $\Gamma \vDash^{n} A\langle \overline{u} \rangle$ for some n such that $\mu = (n, 0)$. Using Lemma 5.3.2 we deduce $\Gamma \vdash^{m} Q$ for some $m \leq n$. Using Lemma 5.3.4 we deduce that $\Gamma \vDash^{\nu} Q$ for some $\nu \leq (m, 0)$. We conclude observing that $\nu \leq (m, 0) \leq (n, 0) = \mu$. $\square$

5.3.3 Normal Forms

𝄢 As in Section 4.3.3 we need to prove that a well typed process is deadlock free. The key lemma of this is Lemma 5.3.11 that we dub *quasi deadlock freedom*. Informally, we want to rearrange the process under analysis in order to put together the creation of some session s with all the subprocesses that start with some communication on s. This way we can try to apply a reduction rule (see Figure 5.3). This is what Lemma 5.3.11 does. However, a process organized as described is not guaranteed to make progress. Indeed, for example all the subprocesses can be waiting for a message leading to a stuck process. Hence, we called the lemma *quasi* deadlock freedom. Deadlock freedom is achieved by taking into account the *coherence* of the session as well. We called the shape of the process mentioned before *proximity normal form* (Definition 5.3.5). In order to obtain a process in such form from a well typed one we require several steps. We introduce additional normal forms (Definitions 5.3.3 and 5.3.4) to describe those processes in the middle of the procedure. We introduce *process contexts* to easily refer to unguarded sub-processes:

$$\textbf{Process context} \quad \mathcal{C}, \mathcal{D} \quad ::= \quad [\,] \mid (s)(\overline{P} \mid \mathcal{C} \mid \overline{Q}) \mid \lceil u \rceil \mathcal{C}$$

Definition 5.3.3 (Choice Normal Form)**.** We say that $P_1 \oplus P_2$ is an *unguarded choice* of P if there exists $\mathcal{C}$ such that $P \preccurlyeq \mathcal{C}[P_1 \oplus P_2]$. We say that P is in *choice normal form* if it has no unguarded choices.

Definition 5.3.4 (Thread Normal Form)**.** A process is in *thread normal form* if it is generated by the grammar below:

$$\begin{array}{rcl} P^{nf}, Q^{nf} & ::= & \lceil u \rceil P^{nf} \mid P^{par} \\ P^{par}, Q^{par} & ::= & (s)(\overline{P^{par}}) \mid P^{th} \\ P^{th} & ::= & \mathsf{done} \mid \mathsf{close}\, u \mid \mathsf{wait}\, u.P \mid u[\mathsf{p}]\pi\{\mathsf{a}_i.P_i\}_{i \in I} \\ & & \mid u[\mathsf{p}]!v.P \mid u[\mathsf{p}]?(x).P \end{array}$$

Intuitively, a process is in *thread normal form* if it consists of an initial prefix of casts followed by a parallel composition of threads, where a thread is either done or a process waiting to perform an input/output action on some channel $u = s[\mathsf{p}]$ for some p. In this latter case, we say that the thread is an s-thread.

Definition 5.3.5 (Proximity Normal Form)**.** We say that P^{nf} is in *proximity normal form* if $P^{nf} = \mathcal{C}[(s)(\overline{P^{th}})]$ for some $\mathcal{C}$, s, $\overline{P^{th}}$ where each P^{th}_i for $i = 1, \dots, h$ is a s-thread.

Lemma 5.3.6. If $\Gamma \vdash^n P$ and $\Gamma \vdash_{\mathsf{ind}} P$, then there exists Q in choice normal form such that $P \Rightarrow Q$ and $\Gamma \vdash^m Q$ for some $m \leq n$.

Proof. By induction on $\Gamma \vdash_{\mathsf{ind}} P$ and by cases on the last rule applied.

Case P is already in choice normal form. We conclude taking $Q \overset{\mathsf{def}}{=} P$ and $m \overset{\mathsf{def}}{=} n$.

Case [TM-CALL]. Then $P = A\langle \overline{u} \rangle$ and $A(\overline{x}) \overset{\triangle}{=} R$. We deduce $\Gamma = \overline{u : S}$, $A : [\overline{S}; n']$ and $\Gamma \vdash_{\mathsf{ind}} R\{\overline{u}/\overline{x}\}$. Moreover, it must be the case that $\Gamma \vdash^{n'} R\{\overline{u}/\overline{x}\}$ and $n' \leq n$ since [TM-CALL] is used in the coinductive judgment as well. Using the induction hypothesis we deduce that there exist Q in choice normal form and $m \leq n'$ such that $R\{\overline{u}/\overline{x}\} \Rightarrow Q$ and $\Gamma \vdash^m Q$. We conclude by observing that $P \Rightarrow Q$ using [RM-STRUCT] and that $m \leq n' \leq n$.

Case [COM-CHOICE]. Then $P = P_1 \oplus P_2$. We deduce $\Gamma \vdash_{\mathsf{ind}} P_k$ with $k \in \{1, 2\}$. Moreover, it must be the case that $\Gamma \vdash^n P_k$ since [TM-CHOICE] is used in the coinductive judgment. Using the induction hypothesis we deduce that there exist Q in choice normal form and $m \leq n$ such that $P_k \Rightarrow Q$ and $\Gamma \vdash^m Q$. We conclude by observing that $P \rightarrow P_k$ by [RM-CHOICE].

Case [TM-CHOICE]. Analogous to the previous case but we consider the premise in which the rank is the same of the conclusion to keep sure that it does not increase.

Case [TM-PAR]. Then $P = (s)(P_1 \mid \cdots \mid P_h)$. We deduce

- $\Gamma = \Gamma_1, \dots, \Gamma_h$
- $\Gamma_i, s[\mathsf{p}_i] : S_i \vdash_{\mathsf{ind}} P_i$ for $i = 1, \dots, h$
- $\prod_{i=1}^{h} \mathsf{p}_i \triangleright S_i$ coherent

Furthermore, it must be the case that $\Gamma_i, s[\mathsf{p}_i] : S_i \vdash^{n_i} P_i$ for $i = 1, \dots, h$ and $n = 1 + \sum_{i=1}^{h} n_i$ since [TM-PAR] is used in the coinductive judgment as well. Using the induction hypothesis we deduce that there exist Q_i in choice normal form and $m_i \leq n_i$ such that $P_i \Rightarrow Q_i$ and $\Gamma_i, s[\mathsf{p}_i] : S_i \vdash^{m_i} Q_i$ for $i = 1, \dots, h$. We conclude by taking $m \overset{\mathsf{def}}{=} 1 + \sum_{i=1}^{h} m_i$ and $Q \overset{\mathsf{def}}{=} (s)(Q_1 \mid \cdots \mid Q_h)$ with one application of [TM-PAR], observing that $m = 1 + \sum_{i=1}^{h} m_i \leq 1 + \sum_{i=1}^{h} n_i = n$ and that $P \Rightarrow Q$ by [RM-PAR].

Case [TM-CAST]. Then $P = \lceil u \rceil P'$. Analogous to the previous case, just simpler. □

Lemma 5.3.7. If $\Gamma \vdash^n P$, then there exists Q in choice normal form such that $P \Rightarrow Q$ and $\Gamma \vdash^m Q$ for some $m \leq n$.

Proof.
Consequence of Lemma 5.3.6 noting that $\Gamma \vdash^n P$ implies $\Gamma \vdash_{\mathsf{ind}} P$. □

Lemma 5.3.8. If $\Gamma \vDash^\mu P$, then there exist Q in choice normal form and $\nu \leq \mu$ such that $P \Rightarrow Q$ and $\Gamma \vDash^\nu Q$.

Proof. By induction on $\Gamma \vDash^\mu P$ and by cases on the last rule applied.

Case [MTM-THREAD]. Then P is a thread. We deduce that

- $\mu = (n, 0)$ for some n
- $\Gamma \vdash^n P$

From Lemma 5.3.7 we deduce that there exist Q and $m \leq n$ such that $P \Rightarrow Q$ and $\Gamma \vdash^m Q$. From Lemma 5.3.4 we deduce $\Gamma \vDash^\nu Q$ for some $\nu \leq (m, 0)$. We conclude observing that $\nu \leq (m, 0) \leq (n, 0) = \mu$.

Case [MTM-CAST]. Then $P = \lceil u \rceil P'$. We deduce that

- $\Gamma = \Delta, u : S$
- $S \leqslant_n T$
- $\mu = \mu' + (n, 0)$
- $\Gamma', u : T \vDash^{\mu'} P'$

Using the induction hypothesis we deduce that there exist Q' and $\nu' \leq \mu'$ such that $P' \Rightarrow Q'$ and $\Gamma', u : T \vDash^{\nu'} Q'$. We conclude with an application of [MTM-CAST] taking $Q \overset{\text{def}}{=} \lceil u \rceil Q'$, $\nu \overset{\text{def}}{=} \nu' + (n, 0)$ and observing that $P \Rightarrow Q$ using [RM-CAST].

Case [MTM-PAR]. Then $P = (s)(P_1 \mid \cdots \mid P_h)$. We deduce

- $\Gamma = \Gamma_1, \dots, \Gamma_h$
- $\mu = \sum_{i=1}^h \mu_i + (0, \| \prod_{i=1}^h \mathsf{p}_i \triangleright S_i \|)$
- $\Gamma_i, s[\mathsf{p}_i] : S_i \vDash^{\mu_i} P_i$ for $i = 1, \dots, h$
- $\prod_{i=1}^h \mathsf{p}_i \triangleright S_i$ coherent

Using the induction hypothesis we deduce that there exist Q_i in choice normal form and $\nu_i \leq \mu_i$ such that $P_i \Rightarrow Q_i$ and $\Gamma_i, s[\mathsf{p}_i] : S_i \vDash^{\nu_i} Q_i$ for $i = 1, \dots, h$. We conclude by taking $\nu \overset{\text{def}}{=} \sum_{i=1}^h \nu_i + (0, \| \prod_{i=1}^h \mathsf{p}_i \triangleright S_i \|)$ and $Q \overset{\text{def}}{=} (s)(Q_1 \mid \cdots \mid Q_h)$ with one application of [MTM-PAR], observing that $\nu = \sum_{i=1}^h \nu_i + (0, \| \prod_{i=1}^h \mathsf{p}_i \triangleright S_i \|) \leq \sum_{i=1}^h \mu_i + (0, \| \prod_{i=1}^h \mathsf{p}_i \triangleright S_i \|) = \mu$ and that $P \Rightarrow Q$ by [RM-PAR]. □

Lemma 5.3.9. If $\Gamma \vdash_{\mathsf{ind}} P$ and P is in choice normal form, then there exists P^{nf} such that $P \preccurlyeq P^{nf}$.

Proof. By induction on $\Gamma \vdash_{\mathsf{ind}} P$ and by cases on the last rule applied.

Cases [TM-CHOICE] *and* [COM-CHOICE]. These cases are impossible from the hypothesis that P is in choice normal form.

Cases [TM-DONE], [TM-WAIT] *and*[TM-CLOSE]. Then P is a thread and is already in thread normal form and we conclude by reflexivity of $\preccurlyeq$.

[TM-CHANNEL-IN], [TM-CHANNEL-OUT], [TM-TAG] *and* [COM-TAG]. Then P is a thread and is already in thread normal form and we conclude by reflexivity of $\preccurlyeq$.

Case [TM-CALL]. Then there exist A, Q, $\overline{u}$ and $\overline{S}$ such that

- $P = A\langle \overline{u} \rangle$

- $A(\overline{x}) \stackrel{\triangle}{=} Q$
- $\Gamma = \overline{u : S}$
- $\overline{u : S} \vdash_{\mathsf{ind}} Q\{\overline{u}/\overline{x}\}$

Using the induction hypothesis on $\overline{u : S} \vdash_{\mathsf{ind}} Q\{\overline{u}/\overline{x}\}$ we deduce that there exists P^{nf} such that $Q\{\overline{u}/\overline{x}\} \preccurlyeq P^{nf}$. We conclude $P \preccurlyeq P^{nf}$ using [SM-CALL] and the transitivity of $\preccurlyeq$.

Case [TM-PAR]. Then there exist s and $P_i, \Gamma_i, S_i, \mathsf{p}_i$ for $i = 1, \dots, h$ such that

- $P = (s)(P_1 \mid \cdots \mid P_h)$
- $\Gamma = \Gamma_1, \dots, \Gamma_h$
- $\Gamma_i, s[\mathsf{p}_i] : S_i \vdash_{\mathsf{ind}} P_i$ for $i = 1, \dots, h$

Using the induction hypothesis on $\Gamma_i, s[\mathsf{p}_i] : S_i \vdash_{\mathsf{ind}} P_i$ we deduce that there exist P_i^{nf} such that $P_i \preccurlyeq P_i^{nf}$ for $i = 1, \dots, h$. By definition of thread normal form, it must be the case that $P_i^{nf} = \lceil \overline{u_i} \rceil P_i^{par}$ for some $\overline{u_i}$ and P_i^{par}. Let $\overline{v_i}$ be the same sequence as $\overline{u_i}$ except that occurrences of $s[\mathsf{p}_i]$ have been removed. We conclude by taking $P^{nf} \stackrel{\mathsf{def}}{=} \lceil \overline{v_1} \dots \overline{v_h} \rceil (s)(P_1^{par} \mid \cdots \mid P_h^{par})$ and using the fact that $\preccurlyeq$ is a pre-congruence and observing that

$$\begin{array}{rcll}
P & = & (s)(P_1 \mid \cdots \mid P_h) & \text{by definition of } P \\
 & \preccurlyeq & (s)(P_1^{nf} \mid \cdots \mid P_h^{nf}) & \text{using the induction hypothesis} \\
 & = & (s)(\lceil \overline{u_1} \rceil P_1^{par} \mid \cdots \mid \lceil \overline{u_h} \rceil P_h^{par}) & \text{by Definition 5.3.4} \\
 & \preccurlyeq & \lceil \overline{v_1} \dots \overline{v_h} \rceil (s)(P_1^{par} \mid \cdots \mid P_h^{par}) & \text{by [SM-CAST-NEW],} \\
 & & & \text{[SM-CAST-SWAP], [SM-PAR-COMM]} \\
 & = & P^{nf} & \text{by definition of } P^{nf}
\end{array}$$

Case [TM-CAST]. Then there exist u, Q, Γ', S and T such that

- $P = \lceil u \rceil Q$
- $\Gamma = \Gamma', u : S$
- $\Gamma', u : T \vdash_{\mathsf{ind}} Q$
- $S \leqslant T$

Using the induction hypothesis on $\Gamma', u : T \vdash_{\mathsf{ind}} Q$ we deduce that there exists Q^{nf} such that $Q \preccurlyeq Q^{nf}$. We conclude by taking $P^{nf} \stackrel{\mathsf{def}}{=} \lceil u \rceil Q^{nf}$ using the fact that $\preccurlyeq$ is a pre-congruence. $\square$

Lemma 5.3.10 (Proximity)**.** If $s \in \mathsf{fn}(P) \setminus \mathsf{bn}(\mathcal{C})$, then $(s)(\mathcal{C}[P] \mid \overline{Q}) \preccurlyeq \mathcal{D}[(s)(P \mid \overline{Q})]$ for some $\mathcal{D}$.

Proof. By induction on the structure of $\mathcal{C}$ and by cases on its shape.

Case $\mathcal{C} = [\ ]$. We conclude by taking $\mathcal{D} \stackrel{\mathsf{def}}{=} [\ ]$ using the reflexivity of $\preccurlyeq$.

Case $\mathcal{C} = (t)(\overline{P'}|\mathcal{C}'|\overline{Q'})$. From the hypothesis $s \in \mathsf{fn}(P)\setminus\mathsf{bn}(\mathcal{C})$ we deduce $s \neq t$ and $s \in \mathsf{fn}(P) \setminus \mathsf{bn}(\mathcal{C}')$. Using the induction hypothesis and [SM-PAR-COMM] we deduce that there exists $\mathcal{D}'$ such that $(s)(\mathcal{C}'[P]|\overline{Q}) \preccurlyeq \mathcal{D}'[(s)(P|\overline{Q})]$. Take $\mathcal{D} \stackrel{\mathsf{def}}{=} (t)(\mathcal{D}' \mid \overline{P'} \mid \overline{Q'})$. We conclude

$$\begin{array}{rcll}
(s)(\mathcal{C}[P] \mid \overline{Q}) & = & (s)((t)(\overline{P'} \mid \mathcal{C}'[P] \mid \overline{Q'}) \mid \overline{Q}) & \text{by definition of } \mathcal{C} \\
& \preccurlyeq & (s)(\overline{Q} \mid (t)(\mathcal{C}'[P] \mid \overline{P'} \mid \overline{Q'})) & \text{by [SM-PAR-COMM]} \\
& \preccurlyeq & (t)((s)(\overline{Q} \mid \mathcal{C}'[P]) \mid \overline{P'} \mid \overline{Q'}) & \text{by [SM-PAR-ASSOC]} \\
& & & \text{and } s \in \mathsf{fn}(\mathcal{C}'[P]) \\
& \preccurlyeq & (t)((s)(\mathcal{C}'[P] \mid \overline{Q}) \mid \overline{P'} \mid \overline{Q'}) & \text{by [SM-PAR-COMM]} \\
& \preccurlyeq & (t)(\mathcal{D}'[(s)(P \mid \overline{Q})] \mid \overline{P'} \mid \overline{Q'}) & \text{by induction hypothesis} \\
& = & \mathcal{D}[(s)(P \mid \overline{Q})] & \text{by definition of } \mathcal{D}
\end{array}$$

Case $\mathcal{C} = \lceil t[\mathsf{p}]\rceil\mathcal{C}'$ *and* $s \neq t$. Using the induction hypothesis we deduce that there exists $\mathcal{D}'$ such that $(s)(\mathcal{C}'[P] \mid \overline{Q}) \preccurlyeq \mathcal{D}'[(s)(P \mid \overline{Q})]$. Take $\mathcal{D} \stackrel{\mathsf{def}}{=} \lceil t[\mathsf{p}]\rceil\mathcal{D}'$. We conclude

$$\begin{array}{rcll}
(s)(\mathcal{C}[P] \mid \overline{Q}) & = & (s)(\lceil t[\mathsf{p}]\rceil\mathcal{C}'[P] \mid \overline{Q}) & \text{by definition of } \mathcal{C} \\
& \preccurlyeq & \lceil t[\mathsf{p}]\rceil(s)(\mathcal{C}'[P] \mid \overline{Q}) & \text{by [SM-CAST-SWAP] and } t \neq s \\
& \preccurlyeq & \lceil t[\mathsf{p}]\rceil\mathcal{D}'[(s)(P \mid \overline{Q})] & \text{using the induction hypothesis} \\
& = & \mathcal{D}[(s)(P \mid \overline{Q})] & \text{by definition of } \mathcal{D}
\end{array}$$

Case $\mathcal{C} = \lceil s[\mathsf{p}]\rceil\mathcal{C}'$. Using the induction hypothesis we deduce that there exists $\mathcal{D}$ such that $(s)(\mathcal{C}'[P] \mid \overline{Q}) \preccurlyeq \mathcal{D}[(s)(P \mid \overline{Q})]$. We conclude

$$\begin{array}{rcll}
(s)(\mathcal{C}[P] \mid \overline{Q}) & = & (s)(\lceil s[\mathsf{p}]\rceil\mathcal{C}'[P] \mid \overline{Q}) & \text{by definition of } \mathcal{C} \\
& \preccurlyeq & (s)(\mathcal{C}'[P] \mid \overline{Q}) & \text{by [SM-CAST-NEW]} \\
& \preccurlyeq & \mathcal{D}[(s)(P \mid \overline{Q})] & \text{using the induction hypothesis}
\end{array}$$

□

Lemma 5.3.11 (Quasi - Deadlock Freedom)**.** If $\emptyset \vDash^{\mu} P^{nf}$, then $P^{nf} = \mathsf{done}$ or $P^{nf} \preccurlyeq Q^{nf}$ for some Q^{nf} in proximity normal form.

Proof. By induction on the derivation of $\emptyset \vDash^{\mu} P^{nf}$ we deduce that P^{nf} consists of $s_1, \ldots, s_h$ sessions and $\sum_{i=1}^{h} k_i - h + 1$ threads where k_i is the number of roles in s_i. The scenarios in which no communication is possible are those in which for each session s_i there are less than k_i s_i-threads. If we assume that for each s_i there are $k_i - 1$ threads, then we obtain

$$\sum_{i=1}^{h} k_i - h + 1 - \sum_{i=1}^{h}(k_i - 1) = \sum_{i=1}^{h} k_i - h + 1 - \sum_{i=1}^{h} k_i + h = 1$$

s_i-thread for some s_i; hence, there exist k_i s_i-threads. In other words, there exist $\mathcal{D}, \mathcal{C}_1, \ldots, \mathcal{C}_{k_i}$ and $P_1^{th}, \ldots, P_{k_i}^{th}$ s_i-threads such that

$$P^{nf} = \mathcal{D}[(s_i)(\mathcal{C}_1[P_1^{th}] \mid \cdots \mid \mathcal{C}_{k_i}[P_{k_i}^{th}])]$$

We conclude

$$\begin{array}{rcl}
P^{nf} & = & \mathcal{D}[(s_i)(\mathcal{C}_1[P_1^{th}] \mid \cdots \mid \mathcal{C}_{k_i}[P_{k_i}^{th}])] \\
& & \hfill \text{by definition of } P^{nf} \\
& \preccurlyeq & \mathcal{D}[\mathcal{D}_1[(s_i)(P_1^{th} \mid \mathcal{C}_2[P_2^{th}] \mid \cdots \mid \mathcal{C}_{k_i}[P_{k_i}^{th}])]] \\
& & \hfill \text{by Lemma 5.3.10} \\
& \preccurlyeq & \mathcal{D}[\mathcal{D}_1[(s_i)(\mathcal{C}_2[P_2^{th}] \mid P_1^{th} \mid \cdots \mid \mathcal{C}_{k_i}[P_{k_i}^{th}])]] \\
& & \hfill \text{by [SM-PAR-COMM]} \\
\ldots & & \\
& \preccurlyeq & \mathcal{D}[\mathcal{D}_1[\mathcal{D}_2[\ldots\mathcal{D}_{k_i}[(s_i)(P_{k_i}^{th} \mid \cdots \mid P_2^{th} \mid P_1^{th})]\ldots]]] \\
& & \hfill \text{for some } \mathcal{D}_2, \ldots, \mathcal{D}_{k_i} \text{ by Lemma 5.3.10} \\
& \stackrel{\text{def}}{=} & Q^{nf}
\end{array}$$

The fact that Q^{nf} is in thread normal form follows from the observation that P^{nf} does not have unguarded casts (it is a closed process in thread normal form) so the pre-congruence rules applied here and in Lemma 5.3.10 do not move casts around. We conclude that Q^{nf} is in proximity normal form by its shape. □

As mentioned at the beginning, Lemma 5.3.11 is dubbed "quasi-deadlock freedom" because it does not say that Q^{nf} reduces. Indeed, a process in proximity normal form is only *ready to communicate* thanks to its shape (see reduction rules). We can prove that a well typed process of such kind actually reduces by observing that [TM-PAR] requires that the involved session is coherent. This result is the key ingredient for proving Lemma 5.3.12.

5.3.4 Soundness

𝄢 The next key result we prove is inspired by the *helpful direction*. Given a well typed process in proximity normal form (see Definition 5.3.5), we prove that there exists a reduct that is well typed with a *strictly smaller* measure (see Lemma 5.3.12). Notably, this result implies deadlock freedom. Then, it is easy to prove that a well typed process is either done or it can reach done in finitely many steps by induction over the measure.

Lemma 5.3.12. If $\Gamma \vDash^{\mu} P^{nf}$ where P^{nf} is in proximity normal form, then there exist Q and $\nu < \mu$ such that $P^{nf} \Rightarrow^{+} Q$ and $\Gamma \vDash^{\nu} Q$.

Proof. From the hypothesis that P^{nf} is in proximity normal form we know that $P^{nf} = \mathcal{C}[(s)(P_1^{th} \mid \cdots \mid P_h^{th})]$ for some $\mathcal{C}$, s and $P_1^{th}, \ldots, P_h^{th}$ s-threads. We reason by induction on $\mathcal{C}$ and by cases on its shape.

Case $\mathcal{C} = [\ ]$. From [MTM-THREAD] and [MTM-PAR] we deduce that there exist $\Gamma_i, S_i, \mathsf{p}_i, n_i$ for $i = 1, \ldots, h$ such that

- $\Gamma = \Gamma_1, \ldots, \Gamma_h$
- $\prod_{i=1}^{h} \mathsf{p}_i \triangleright S_i$ coherent
- $\mu = (\sum_{i=1}^{h} n_i, \| \prod_{i=1}^{h} \mathsf{p}_i \triangleright S_i \|)$
- $\Gamma_i, s[\mathsf{p}_i] : S_i \vdash^{n_i} P_i^{th}$ for $i = 1, \ldots, h$

From the hypothesis that $\prod_{i=1}^{h} \mathsf{p}_i \triangleright S_i$ coherent we deduce $\prod_{i=1}^{h} \mathsf{p}_i \triangleright S_i \overset{?\checkmark}{\Longrightarrow}$. We now reason on the rank of the session and on the shape of S_i. For the sake of simplicity, we implicitly apply [S-PAR-COMM] at process level.

If $\| \prod_{i=1}^{h} \mathsf{p}_i \triangleright S_i \| = 1$, then $\prod_{i=1}^{h} \mathsf{p}_i \triangleright S_i \xrightarrow{?\checkmark}$ using [LM-TERMINATE].

- *Case* $S_1 = ?\mathsf{end}$ *and* $S_j = !\mathsf{end}$ *for* $j = 2, \ldots, h$. Then
  - $\Gamma_j = \emptyset$ and $P_j^{th} = \mathsf{close}\, s[\mathsf{p}_j]$ for $j = 2, \ldots, h$
  - $P_1^{th} = \mathsf{wait}\, s[\mathsf{p}_1].Q$
  - $\Gamma_1 \vdash^{n_1} Q$

  From Lemma 5.3.4 we deduce that $\Gamma_1 \vDash^{\nu} Q$ for some $\nu \leq (n_1, 0)$. We conclude observing that $P^{nf} \rightarrow Q$ by [RM-SIGNAL] and that $\nu \leq (n_1, 0) < (\sum_{i=1}^{h} n_i, \| \prod_{i=1}^{h} \mathsf{p}_i \triangleright S_i \|) = \mu$.

If $\| \prod_{i=1}^{h} \mathsf{p}_i \triangleright S_i \| > 1$, then $\prod_{i=1}^{h} \mathsf{p}_i \triangleright S_i \xrightarrow{\tau} \ldots \xrightarrow{?\checkmark}$ first using [LM-TAU] and [LM-SYNC]. Observe that [LM-PICK] is never used since we are considering the minimum reduction sequence; a synchronization through [LM-PICK] and [LM-SYNC] would lead to a longer reduction. T hen $S_1 \xrightarrow{\mathsf{p}_1 \triangleright \mathsf{p}_2 ! \mathsf{m}_k}$ and $S_2 \xrightarrow{\mathsf{p}_2 \triangleright \mathsf{p}_1 ? \mathsf{m}_k}$ for some m_k or $S_1 \xrightarrow{\mathsf{p}_1 \triangleright \mathsf{p}_2 ! S}$ and $S_2 \xrightarrow{\mathsf{p}_1 \triangleright \mathsf{p}_2 ? S}$.

- *Case* $S_1 = \sum_{i\in I} \mathsf{p}_2!\mathsf{m}_i.S'_i$ *and* $S_2 = \sum_{j\in J} \mathsf{p}_1?\mathsf{m}_j.T_j$ *with* $k \in I$. From the hypothesis that $\prod_{i=1}^{h} \mathsf{p}_i \triangleright S_i$ coherent we deduce $I \subseteq J$. From Definition 2.3.2 we deduce $\prod_{i=3}^{h} \mathsf{p}_i \triangleright S_i \mid \mathsf{p}_1 \triangleright S'_k \mid \mathsf{p}_2 \triangleright T_k$ coherent and from [TM-TAG] we deduce that

  - $P_1^{th} = s[\mathsf{p}_1][\mathsf{p}_2]!\{\mathsf{m}_i.P'_i\}_{i\in I}$
  - $P_2^{th} = s[\mathsf{p}_2][\mathsf{p}_1]?\{\mathsf{m}_j.Q_j\}_{j\in J}$
  - $\Gamma_1, s[\mathsf{p}_1] : S'_i \vdash^{n_1} P'_i$ for all $i \in I$
  - $\Gamma_2, s[\mathsf{p}_2] : T_j \vdash^{n_2} Q_j$ for all $j \in J$

  Let $Q \overset{\mathsf{def}}{=} (s)(P'_k \mid Q_k \mid P_3 \mid \cdots \mid P_h)$ and observe that $P^{nf} \Rightarrow^{+} Q$ by [RM-PICK] and [RM-TAG]. From Lemma 5.3.4 we deduce that there exist $\mu_1 \leq (n_1, 0), \mu_2 \leq (n_2, 0)$ such that

  - $\Gamma_1, s[\mathsf{p}_1] : S'_k \vDash^{\mu_1} P'_k$
  - $\Gamma_2, s[\mathsf{p}_2] : T_k \vDash^{\mu_2} Q_k$

  Let $\nu \overset{\mathsf{def}}{=} \mu_1 + \mu_2 + (\sum_{i=3}^{h} n_i, \|\mathsf{p}_1 \triangleright S'_k \mid \mathsf{p}_2 \triangleright T_k \mid \prod_{i=3}^{h} \mathsf{p}_i \triangleright S_i\|)$. We conclude with one application of [MTM-PAR] observing that

$$\begin{array}{rcll}
\nu & = & \mu_1 + \mu_2 + & \\
 & & (\sum_{i=3}^{h} n_i, \|\mathsf{p}_1 \triangleright S'_k \mid \mathsf{p}_2 \triangleright T_k \mid \prod_{i=3}^{h} \mathsf{p}_i \triangleright S_i\|) & \text{by def. of } \nu \\
 & \leq & (\sum_{i=1}^{h} n_i, \|\mathsf{p}_1 \triangleright S'_k \mid \mathsf{p}_2 \triangleright T_k \mid \prod_{i=3}^{h} \mathsf{p}_i \triangleright S_i\|) & \text{by Lemma 5.3.4} \\
 & < & (\sum_{i=1}^{h} n_i, \| \prod_{i=1}^{h} \mathsf{p}_i \triangleright S_i\|) & \text{before } \rightarrow \\
 & = & \mu &
\end{array}$$

- *Case* $S_1 = \mathsf{p}_2!S.T_1$ *and* $S_2 = \mathsf{p}_1?S.T_2$.

From the hypothesis that $\prod_{i=1}^{h} \mathsf{p}_i \triangleright S_i$ coherent and Definition 2.3.2 we deduce $\mathsf{p}_1 \triangleright T_1 \mid \mathsf{p}_2 \triangleright T_2 \mid \prod_{i=3}^{h} \mathsf{p}_i \triangleright S_i$ coherent and from [TM-CHANNEL-OUT] and [TM-CHANNEL-IN] we deduce that

- $P_1^{th} = s[\mathsf{p}_1][\mathsf{p}_2]!u.P'_1$
- $P_2^{th} = s[\mathsf{p}_2][\mathsf{p}_1]?(x).P'_2$
- $\Gamma_1, s[\mathsf{p}_1] : T_1 \vdash^{n_1} P'_1$
- $\Gamma_2, s[\mathsf{p}_2] : T_2, x : S \vdash^{n_2} P'_2$

Let $Q \overset{\text{def}}{=} (s)(P'_1 \,|\, P'_2\{u/x\} \,|\, P^{th}_3 \,|\, \cdots \,|\, P^{th}_h)$ and observe that $P^{nf} \to Q$ by [RM-CHANNEL]. Using Lemma 5.3.1 we deduce $\Gamma_2, s[\mathsf{p}_2] : T_2, u : S \vdash^{n_2} P'_2\{u/x\}$ and from Lemma 5.3.4 we deduce that there exist $\mu_1 \leq (n_1, 0), \mu_2 \leq (n_2, 0)$ such that

- $\Gamma_1, s[\mathsf{p}_1] : T_1 \vDash^{\mu_1} P'_1$
- $\Gamma_2, s[\mathsf{p}_2] : T_2, u : S \vDash^{\mu_2} P'_2\{u/x\}$

Let $\nu \overset{\text{def}}{=} \mu_1 + \mu_2 + (\sum_{i=3}^h n_i, \|\mathsf{p}_1 \triangleright T_1 \mid \mathsf{p}_2 \triangleright T_2 \mid \prod_{i=3}^h \mathsf{p}_i \triangleright S_i\|)$. We conclude with one application of [MTM-PAR] observing that

$$\begin{array}{rll} \nu & = & \mu_1 + \mu_2 + \\ & & (\sum_{i=3}^h n_i, \|\mathsf{p}_1 \triangleright T_1 \mid \mathsf{p}_2 \triangleright T_2 \mid \prod_{i=3}^h \mathsf{p}_i \triangleright S_i\|) \quad \text{by definition of } \nu \\ & \leq & (\sum_{i=1}^h n_i, \|\mathsf{p}_1 \triangleright T_1 \mid \mathsf{p}_2 \triangleright T_2 \mid \prod_{i=3}^h \mathsf{p}_i \triangleright S_i\|) \quad \text{by Lemma 5.3.4} \\ & < & (\sum_{i=1}^h n_i, \| \prod_{i=1}^h \mathsf{p}_i \triangleright S_i\|) \quad \text{before reductions} \\ & = & \mu \end{array}$$

Case $\mathcal{C} = (t)(\overline{P^{par}} \,|\, \mathcal{D} \,|\, \overline{Q^{par}})$. Let $R^{nf} \overset{\text{def}}{=} \mathcal{D}[(s)(P^{th}_1 \,|\cdots|\, P^{th}_h)]$ and observe that R^{nf} is in proximity normal form. From [MTM-PAR] we deduce that there exist $\Gamma_i, S_i, \mu_i, \mathsf{p}_i$ for $i = 1, \dots, h$ and $k \leq h$ such that

- $\Gamma = \Gamma_1, \dots, \Gamma_h$
- $\Gamma_1, t[\mathsf{p}_1] : S_1 \vDash^{\mu_1} R^{nf}$
- $\Gamma_i, t[\mathsf{p}_i] : S_i \vDash^{\mu_i} P^{par}_i$ for $i = 1, \dots, k$
- $\Gamma_i, t[\mathsf{p}_i] : S_i \vDash^{\mu_i} Q^{par}_i$ for $i = k+1, \dots, h$
- $\mu = \sum_{i=1}^h \mu_i + (0, \| \prod_{i=1}^h \mathsf{p}_i \triangleright S_i\|)$

Using the induction hypothesis on $\Gamma_1, t[\mathsf{p}_1] : S_1 \vDash^{\mu_1} R^{nf}$ we deduce that there exists Q' and $\nu' < \mu_1$ such that

- $R^{nf} \Rightarrow^+ Q'$
- $\Gamma_1, t[\mathsf{p}_1] : S_1 \vDash^{\nu'} Q'$

We conclude taking $Q \overset{\text{def}}{=} (t)(Q' \mid \overline{P^{par}} \mid \overline{Q^{par}})$ and

$$\nu \overset{\text{def}}{=} \nu' + \sum_{i=2}^{h} \mu_i + (0, \| \prod_{i=1}^{h} \mathsf{p}_i \triangleright S_i\|)$$

and observing that $\nu < \mu$ and $P^{nf} \Rightarrow^+ Q$ by [RM-PAR].

Case $\mathcal{C} = \lceil t[\mathsf{q}] \rceil \mathcal{D}$. Observe that $t \neq s$. Let $R^{nf} \overset{\mathsf{def}}{=} \mathcal{D}[(s)(P_1^{th} \mid \cdots \mid P_h^{th})]$ and note that R^{nf} is in proximity normal form. From [MTM-CAST] we deduce that there exists Δ, μ', S, T, m_t such that

- $\Gamma = \Delta, t[\mathsf{q}] : S$
- $S \leqslant_{m_t} T$
- $\mu = \mu' + (m_t, 0)$
- $\Delta, t[\mathsf{q}] : T \vDash^{\mu'} R^{nf}$

Using the induction hypothesis on $\Delta, t[\mathsf{q}] : T \vDash^{\mu'} R^{nf}$ we deduce that there exist Q' and $\nu' < \mu'$ such that $R^{nf} \Rightarrow^+ Q'$ and $\Delta, t[\mathsf{q}] : T \vDash^{\nu'} Q'$. We conclude taking $Q \overset{\mathsf{def}}{=} \lceil t[\mathsf{q}] \rceil Q'$ and $\nu \overset{\mathsf{def}}{=} \nu' + (m_t, 0)$ and observing that $\nu < \mu$ and $P^{nf} \Rightarrow^+ Q$ by [RM-CAST]. □

Lemma 5.3.13. If $\emptyset \vDash^{\mu} P$, then either $P \preccurlyeq \mathsf{done}$ or $P \Rightarrow^+ Q$ and $\emptyset \vDash^{\nu} Q$ for some Q and $\nu < \mu$.

Proof. Using Lemma 5.3.8 we deduce that there exist P' in choice normal form such that $P \Rightarrow P'$ and $\emptyset \vDash^{\mu'} P'$ and $\mu' \leq \mu$. By Lemma 5.3.4 we deduce $\emptyset \vdash P'$. Using Lemma 5.3.9 we deduce that there exist P^{nf} such that $P' \preccurlyeq P^{nf}$.

If $P^{nf} = \mathsf{done}$ there is nothing left to prove.

If $P^{nf} \neq \mathsf{done}$, by Lemma 5.3.11 we deduce $P^{nf} \preccurlyeq Q^{nf}$ for some Q^{nf} in proximity normal form. From Lemma 5.3.5 we deduce $\emptyset \vDash^{\mu''} Q^{nf}$ for some $\mu'' \leq \mu'$. Using Lemma 5.3.12 we conclude that $Q^{nf} \Rightarrow^+ Q$ and $\emptyset \vDash^{\nu} Q$ for some Q and $\nu < \mu'' \leq \mu' \leq \mu$. □

Lemma 5.3.14. If $\emptyset \vdash^{n} P$, then either $P \preccurlyeq \mathsf{done}$ or $P \Rightarrow^+ \mathsf{done}$.

Proof. From Lemma 5.3.4 we deduce that there exists $\mu \leq (n, 0)$ such that $\emptyset \vDash^{\mu} P$. We proceed doing an induction on the lexicographically ordered pair μ. From Lemma 5.3.13 we deduce either $P \preccurlyeq \mathsf{done}$ or $P \Rightarrow^+ Q$ and $\emptyset \vDash^{\nu} Q$ for some $\nu < \mu$. In the first case there is nothing left to prove. In the second case we use the induction hypothesis to deduce that either $Q \preccurlyeq \mathsf{done}$ or $Q \Rightarrow^+ \mathsf{done}$. We conclude using either [RM-STRUCT] or the transitivity of $\Rightarrow^+$, respectively. □

Proof of Theorem 5.2.1.
Immediate consequence of Lemmas 5.3.3 and 5.3.14. □

5.4 Related Work

We conclude the chapter by relating the type systems in Chapters 4 and 5 to others that can be found in the literature. In particular, we mainly recall the references we pointed out in Section 1.4.

Termination of Binary Sessions. Lindley and Morris [2016] define a type system for a functional language with session primitives and recursive session types that is strongly normalizing. That is, a well-typed program along with all the sessions it creates is guaranteed to terminate. This strong result is due to the fact that the type language is equipped with least and greatest fixed point operators that are required to match each other by duality. Notably, strong normalization is stronger than fair termination. As an example, Examples 4.1.1 and 5.1.1 are fairly terminating but not strongly terminating as the buyer can add arbitrarily many, possibly infinitely many, items to the cart.

Liveness Properties in the π-Calculus. Kobayashi [2002a], Padovani [2014] define a behavioral type system that guarantees lock freedom in the π-calculus. . These works annotate types with numbers representing finite upper bounds to the number of interactions needed to unblock a particular input/output action. For this reason, none of our key examples (Examples 5.1.1 to 5.1.3) is in the scope of these analysis techniques. Kobayashi and Sangiorgi [2010] show how to enforce lock freedom by combining deadlock freedom and termination. Our work can be seen as a generalization of this approach whereby we enforce lock freedom by combining deadlock freedom (through a mostly conventional session type system) and *fair* termination. Since fair termination is coarser than termination, the family of programs for which lock freedom can be proved is larger as well.

Deadlock Freedom. Our type system enforces deadlock freedom essentially thanks to the shape of the rule [tm-par] ([tb-par] in Chapter 4) which is inspired to the cut rule of linear logic. This rule has been applied to session type systems for binary sessions [Wadler, 2014, Caires et al., 2016, Lindley and Morris, 2016] and subsequently extended to multiparty sessions [Carbone et al., 2016, 2017]. In the latter case, the rule – dubbed *multiparty cut* – requires a coherence condition among cut types establishing that the session types followed by the single participants adhere to a so-called global type describing the multiparty session as a whole. The rule [tm-par] adopts the same schema, except that the coherence condition is stronger to entail fair session termination. The key principle of these formulations of the cut rule as a typing rule for parallel processes is to impose a tree-like network topology, whereby two parallel processes can share at most one channel. In the multiparty case, cyclic network topologies can be modeled within each session (Example 5.1.3) since coherence implies deadlock freedom.

Having a single construct that merges session restriction and parallel composition allows for a simple formulation of the typing rules so that dealock freedom is easily guaranteed. However, many session calculi separate these two forms in line with the original presentation of the π-calculus. We think that our type system can be easily reformulated to support distinct session restriction and parallel composition by means of hypersequents [Kokke et al., 2018, 2019].

A more liberal version of the cut rule, named multi-cut and inspired to Gentzen's "mix" rule, is considered by [Abramsky et al., 1996] enabling processes to share more than one channel. In this setting, deadlock freedom is lost but can be recovered by means of a richer type structure that keeps track of the dependencies between different channels. This approach has been pioneered by Kobayashi [2002a, 2006] for the π-calculus and later on refined by Padovani [2014]. Other approaches to ensure deadlock freedom based on *dependency/connectivity graphs* that capture the network topology implemented by processes have been studied by Carbone and Debois [2010], Kobayashi and Laneve [2017], de'Liguoro and Padovani [2018], Jacobs et al. [2022].

Liveness Properties of Multiparty Sessions. In Scalas and Yoshida [2019] the authors point out that the coarsest liveness property in the hierarchy of liveness properties that they take into account, which is the one more closely related to fair termination, cannot be enforced by their type system. In part, this is due to the fact that their type system relies on a standard subtyping relation for session types [Gay and Hole, 2005] instead of fair subtyping [Padovani, 2013, 2016]. As we have seen in Chapter 4, even for single-session programs the mere adoption of fair subtyping is not enough and it is necessary to meet additional requirements. van Glabbeek et al. [2021] propose a type system for multiparty sessions that ensures progress and is not only sound but also complete. The fairness assumption they make – called *justness* – is substantially weaker than our own (Definition 2.2.3) and such that the unfair runs are those in which some interactions between participants are systematically discriminated in favor of other interactions involving a disjoint set of independent participants. For this reason, their progress property is in between the two more restrictive liveness predicates of Scalas and Yoshida [2019] and can only be guaranteed when it is independent of the behavior of the other participants of the same session.

Chapter 6

Linear Logic Based Approach

In this chapter we propose a type system for $\pi\mathsf{LIN}$, a linear π-calculus with (co)recursive types, such that well-typed processes are *fairly terminating*. Our type system is a conservative extension of $\mu\mathsf{MALL}^\infty$ [Baelde et al., 2016, Doumane, 2017, Baelde et al., 2022], the infinitary proof system for the multiplicative additive fragment of linear logic with least and greatest fixed points. In fact, the modifications we make to $\mu\mathsf{MALL}^\infty$ are remarkably small: we add one (standard) rule to deal with *non-deterministic choices*, those performed autonomously by a process, and we relax the validity condition on $\mu\mathsf{MALL}^\infty$ proofs so that it only considers the "fair behaviors" of the program it represents. The fact that there is such a close correspondence between the typing rules of $\pi\mathsf{LIN}$ and the inference rules of $\mu\mathsf{MALL}^\infty$ is not entirely surprising. After all, there have been plenty of works investigating the relationship between π-calculus terms and linear logic proofs, from those of Abramsky [1994], Bellin and Scott [1994] to those on the interpretation of linear logic formulas as session types [DeYoung et al., 2012, Wadler, 2014, Caires et al., 2016, Lindley and Morris, 2016, Rocha and Caires, 2021, Qian et al., 2021].

Nonetheless, we think that the connection between $\pi\mathsf{LIN}$ and $\mu\mathsf{MALL}^\infty$ stands out for two reasons. First, $\pi\mathsf{LIN}$ is conceptually simpler and more general than the session-based calculi that can be encoded in it. In particular, all the session calculi based on linear logic rely on an asymmetric interpretation of the multiplicative connectives $\otimes$ and $⅋$ so that $\varphi \otimes \psi$ (respectively, $\varphi ⅋ \psi$) is the type of a session endpoint used for sending (respectively, receiving) a message of type φ and then used according to ψ. In our setting, the connectives $\otimes$ and $⅋$ retain their symmetry since we interpret $\varphi \otimes \psi$ and $\varphi ⅋ \psi$ formulas as the output/input of pairs, in the same spirit of the original encoding of linear logic proofs proposed by Bellin and Scott [1994]. This interpretation gives $\pi\mathsf{LIN}$ the ability of modeling *bifurcating protocols* of which binary sessions are just a special case. The second reason why $\pi\mathsf{LIN}$ and $\mu\mathsf{MALL}^\infty$ get along has to do with the cut elimination result for $\mu\mathsf{MALL}^\infty$. In finitary proof systems for linear logic, cut elimination may proceed by removing *topmost cuts*. In $\mu\mathsf{MALL}^\infty$ there is no such notion as a topmost cut since $\mu\mathsf{MALL}^\infty$ proofs may be infinite. As a consequence, the cut elimination result for $\mu\mathsf{MALL}^\infty$ is proved by eliminating *bottom-most cuts* [Baelde et al., 2022]. This strategy fits perfectly with the reduction semantics of $\pi\mathsf{LIN}$ – and that of any other conventional process calculus, for that matter – whereby reduction rules act only on the exposed (*i.e.* unguarded) part of processes but not behind prefixes. As a result, the reduction semantics of $\pi\mathsf{LIN}$ is completely ordinary, unlike other logically-inspired process calculi that incorporate commuting conversions [Wadler, 2014, Lindley and Morris, 2016], perform reductions behind prefixes [Qian et al., 2021] or swap prefixes [Bellin and Scott, 1994].

In Chapters 4 and 5 we have proposed a type system ensuring the fair termination of binary/multiparty sessions. In the present chapter we achieve the same objective using a more basic process calculus and exploiting its strong logical connection with $\mu\mathsf{MALL}^\infty$. In fact, the soundness proof of our type system piggybacks on the cut elimination property of $\mu\mathsf{MALL}^\infty$. Other session typed calculi based on linear logic with fixed points have been studied by Lindley and Morris [2016] and Derakhshan and Pfenning [2019], Derakhshan [2021]. The type systems described in these works respectively guarantee termination and strong progress, whereas our type system guarantees fair termination which is somewhat in between these properties. Overall, our type system seems to hit a sweet spot: on the one hand, it is deeply rooted in linear logic and yet it can deal with common communication patterns (like the buyer/seller interaction described above) that admit potentially infinite executions and therefore are out of scope of other logic-inspired type systems; on the other hand, it guarantees lock freedom [Kobayashi, 2002a, Padovani, 2014], strong progress [Derakhshan and Pfenning, 2019, Theorem 12.3] and also termination, under a suitable fairness assumption.

The chapter is organized as follows. In Section 6.1 we show the definition of the types bu relying on the formulas of linear logic. Notably, these contain least/greatest fixed points. In Section 6.2 we present syntax and semantics of $\pi\mathsf{LIN}$. Section 6.3 introduces the type system for $\pi\mathsf{LIN}$ as well as the validity conditions. As usual, we dedicate Section 6.4 to detail the soundness proof of the type system and Section 6.6 to discuss about related works. Finally, we are now able to make a comparison between the present type system and those presented in Chapters 4 and 5. We make such comparison by examples in Section 6.5.

6.1 Types and Formulas

The types of $\pi\mathsf{LIN}$ are built using the multiplicative additive fragment of linear logic enriched with least and greatest fixed points. In this section we specify the syntax of types along with all the auxiliary notions that are needed to present the type system and prove its soundness.

Definition 6.1.1 (Pre-formula)**.** The syntax of pre-formulas relies on an infinite set of *propositional variables* ranged over by X and Y and is defined by the grammar below:

$$\varphi, \psi \quad ::= \quad \mathbf{0} \mid \top \mid \mathbf{1} \mid \bot \mid \varphi \oplus \psi \mid \varphi \mathbin{\&} \psi \mid \varphi \otimes \psi \mid \varphi \parr \psi \mid \mu X.\varphi \mid \nu X.\varphi \mid X$$

As usual, μ and ν are the binders of propositional variables and the notions of free and bound variables are defined accordingly. We assume that the body of fixed points extends as much as possible to the right of a pre-formula, so $\mu X.X \oplus \mathbf{1}$ means $\mu X.(X \oplus \mathbf{1})$ and not $(\mu X.X) \oplus \mathbf{1}$. We write $\{\varphi/X\}$ for the capture-avoiding substitution of all free occurrences of X with φ.

Definition 6.1.2 (Dual of a formula)**.** We write $\varphi^\perp$ for the *dual* of φ, which is the involution defined by the equations

$$\begin{array}{lll} \mathbf{0}^\perp = \top & (\varphi \oplus \psi)^\perp = \varphi^\perp \mathbin{\&} \psi^\perp & (\mu X.\varphi)^\perp = \nu X.\varphi^\perp \\ \mathbf{1}^\perp = \bot & (\varphi \otimes \psi)^\perp = \varphi^\perp \parr \psi^\perp & X^\perp = X \end{array}$$

Definition 6.1.3 (Formula)**.** A *formula* is a closed pre-formula.

In the context of $\pi\mathsf{LIN}$, formulas describe how linear channels are used. Positive formulas (those built with the constants $\mathbf{0}$ and $\mathbf{1}$, the connectives $\oplus$ and $\otimes$ and the least fixed point) indicate output operations whereas negative formulas (the remaining forms) indicate input operations. The formulas $\varphi \oplus \psi$ and $\varphi \mathbin{\&} \psi$ describe a linear channel used for sending/receiving a tagged channel of type φ or ψ. The tag (either in_1 or in_2) distinguishes between the two possibilities. The formulas $\varphi \otimes \psi$ and $\varphi \parr \psi$ describe a linear channel used for sending/receiving a pair of channels of type φ and ψ; $\mu X.\varphi$ and $\nu X.\varphi$ describe a linear channel used for sending/receiving a channel of type $\varphi\{\mu X.\varphi/X\}$ or $\varphi\{\nu X.\varphi/X\}$ respectively. The constants $\mathbf{1}$ and $\bot$ describe a linear channel used for sending/receiving the unit. Finally, the constants $\mathbf{0}$ and $\top$ respectively describe channels on which nothing can be sent and from which nothing can be received.

Example 6.1.1. *Looking at the structure of* ***buyer*** *and* ***seller*** *in Example 2.1.1, we can make an educated guess on the type of the channel they use. Indeed, we see that it is used according to* $\varphi \stackrel{\text{def}}{=} \mu X.X \oplus \mathbf{1}$ *by* ***buyer*** *and according to* $\psi \stackrel{\text{def}}{=} \nu X.X \mathbin{\&} \bot$ *in* ***seller****. Note that* $\varphi = \psi^\bot$*, suggesting that* ***seller*** *and* ***seller*** *may interact correctly when connected.* ⌟

Definition 6.1.4 (Subformula ordering)**.** We write $\preceq$ for the *subformula ordering*, that is the least partial order such that $\varphi \preceq \psi$ if φ is a subformula of ψ.

For example, consider $\varphi \stackrel{\text{def}}{=} \mu X.\nu Y.X \oplus Y$ and $\psi \stackrel{\text{def}}{=} \nu Y.\varphi \oplus Y$. Then we have $\varphi \preceq \psi$ and $\psi \not\preceq \varphi$. When Φ is a set of formulas, we write $\mathsf{min}\,\Phi$ for its $\preceq$-minimum formula if it is defined. Occasionally we let $\star$ stand for an arbitrary binary connective $\oplus$, $\otimes$, $\&$, or $⅋$ and σ stand for an arbitrary fixed point operator μ or ν.

When two $\pi\mathsf{LIN}$ processes interact on some channel x, they may exchange other channels on which their interaction continues. We can think of these subsequent interactions stemming from a shared channel x as being part of the same conversation (the literature on *sessions* [Honda, 1993, Hüttel et al., 2016] builds on this idea [Kobayashi, 2002b, Dardha et al., 2017]). The soundness proof of the type system is heavily based on the proof of the cut elimination property of $\mu\mathsf{MALL}^\infty$, which relies on the ability to uniquely identify the types of the channels that belong to the same conversation and to trace conversations within typing derivations. Following the literature on $\mu\mathsf{MALL}^\infty$ [Baelde et al., 2016, Doumane, 2017, Baelde et al., 2022], we annotate formulas with addresses. We assume an infinite set $\mathcal{A}$ of *atomic addresses*, $\mathcal{A}^\bot$ being the set of their duals such that $\mathcal{A} \cap \mathcal{A}^\bot = \emptyset$ and $\mathcal{A}^{\bot\bot} = \mathcal{A}$. We use a and b to range over elements of $\mathcal{A} \cup \mathcal{A}^\bot$.

Definition 6.1.5 (Address)**.** An *address* is a string aw where $w \in \{i, l, r\}^*$. The dual of an address is defined as $(aw)^\bot = a^\bot w$.

We use α and β to range over addresses, we write $\leq$ for the prefix relation on addresses and we say that α and β are *disjoint* if $\alpha \not\leq \beta$ and $\beta \not\leq \alpha$.

Definition 6.1.6 (Type)**.** A *type* is a formula φ paired with an address α written φ_α.

We use S and T to range over types and we extend to types several operations defined on formulas: we use logical connectives to compose types so that $\varphi_{\alpha l} \star \psi_{\alpha r} \overset{\mathsf{def}}{=} (\varphi \star \psi)_\alpha$ and $\sigma X.\varphi_{\alpha i} \overset{\mathsf{def}}{=} (\sigma X.\varphi)_\alpha$; the dual of a type is obtained by dualizing both its formula and its address, that is $(\varphi_\alpha)^\perp \overset{\mathsf{def}}{=} \varphi^\perp_{\alpha^\perp}$; type substitution preserves the address in the type within which the substitution occurs, but forgets the address of the type being substituted, that is $\varphi_\alpha\{\psi_\beta/X\} \overset{\mathsf{def}}{=} \varphi\{\psi/X\}_\alpha$.

We often omit the address of constants (which represent terminated conversations) and we write $\overline{S}$ for the formula obtained by forgetting the address of S. Finally, we write $\rightsquigarrow$ for the least reflexive relation on types such that $S_1 \star S_2 \rightsquigarrow S_i$ and $\sigma X.S \rightsquigarrow S\{\sigma X.S/X\}$.

Example 6.1.2. *Consider once again the formula $\varphi \overset{\mathsf{def}}{=} \mu X.X \oplus \mathbf{1}$ that describes the behavior of **buyer** (Example 6.1.1) and let a be an arbitrary atomic address. We have*

$$\varphi_a \rightsquigarrow (\varphi \oplus \mathbf{1})_{ai} \rightsquigarrow \varphi_{ail} \rightsquigarrow (\varphi \oplus \mathbf{1})_{aili} \rightsquigarrow \mathbf{1}_{ailir}$$

where the fact that the types in this sequence all share a common non-empty prefix 'a' indicates that they belong to the same conversation. Note how the symbols i, l and r composing an address indicate the step taken in the syntax tree of types for making a move in this sequence: i means "inside", when a fixed point operator is unfolded, whereas l and r mean "left" and "right", when the corresponding branch of a connective is selected. ⌟

6.2 Calculus

In this section we define syntax and reduction semantics of $\pi\mathsf{LIN}$, a variant of the linear π-calculus [Kobayashi et al., 1999] in which all channels are meant to be used for *exactly* one communication. The calculus supports (co)recursive data types built using units, pairs and disjoint sums. These data types are known to be the essential ingredients for the encoding of sessions in the linear π-calculus [Kobayashi, 2002b, Dardha et al., 2017, Scalas et al., 2017].

6.2.1 Syntax of Processes

We assume an infinite set of *channels* ranged over by x, y and z.

$P, Q ::= x \leftrightarrow y$	link	$\mid (x)(P \mid Q)$	comp	
$\mid\ \mathsf{case}\, x\{\}$	empty in	$\mid P \oplus Q$	choice	
$\mid\ x().P$	unit in	$\mid \overline{x}()$	unit out	
$\mid\ x(z, y).P$	pair in	$\mid \overline{x}(z, y)(P \mid Q)$	pair out	
$\mid\ \mathsf{case}\, x(y)\{P, Q\}$	sum in	$\mid \mathsf{in}_i\, \overline{x}(y).P$	sum out	$i \in \{1, 2\}$
$\mid\ \mathsf{corec}\, x(y).P$	corec	$\mid \mathsf{rec}\, \overline{x}(y).P$	rec	

Figure 6.1: Syntax of $\pi\mathsf{LIN}$

$\pi\mathsf{LIN}$ processes are coinductively generated by the productions of the grammar shown in Figure 6.1. A *link* $x \leftrightarrow y$ acts as a *linear forwarder* Gardner et al. [2007] that forwards a single message either from x to y or from y to x. The uncertainty in the direction of the message is resolved once the term is typed and the polarity of the types of x and y is fixed (as we will show in Section 6.3). The term $\mathsf{case}\, x\{\}$ represents a process that receives an empty message from x and then fails. This form is only useful in the metatheory: the type system guarantees that well-typed processes never fail, since it is not possible to send empty messages. The term $\overline{x}()$ models a process that sends the unit on x, effectively indicating that the interaction is terminated, whereas $x().P$ models a process that receives the unit from x and then continues as P. The term $\overline{x}(y, z)(P \mid Q)$ models a process that creates two new channels y and z, sends them in a pair on channel x and then forks into two parallel processes P and Q. Dually, $x(y, z).P$ models a process that receives a pair containing two channels y and z from channel x and then continues as P. The term $\mathsf{in}_i\, \overline{x}(y).P$ models a process that creates a new channel y and sends $\mathsf{in}_i(y)$ (that is, the i-th injection of y in a disjoint sum) on x. Dually, $\mathsf{case}\, x(y)\{P_1, P_2\}$ receives a disjoint sum from channel x and continues as either P_1 or P_2 depending on the tag in_i it has been built with. For clarity, in some examples we will use more descriptive labels such as add and pay instead of in_1 and in_2. The terms $\mathsf{rec}\, \overline{x}(y).P$ and $\mathsf{corec}\, x(y).P$ model processes that respectively send and receive a new channel y and then continue as P. They do not contribute operationally to the interaction being modeled, but they indicate the points in a program where (co)recursive types are unfolded. A term $(x)(P \mid Q)$ denotes the parallel composition of two processes P and Q that interact through the fresh channel x. Finally, the term $P \oplus Q$ models a non-deterministic choice between two behaviors P and Q.

$\pi\mathsf{LIN}$ binders are easily recognizable because they enclose channel names in round parentheses. Note that all outputs are in fact *bound outputs*. The output of free channels can be modeled by combining bound outputs with links [Lindley and Morris, 2016]. For example, the output $\overline{x}\langle y, z\rangle$ of a pair of free channels y and z can be modeled as the term $\overline{x}(y', z')(y \leftrightarrow y' \mid z \leftrightarrow z')$. We identify processes modulo renaming of bound names, we write $\mathsf{fn}(P)$ for the set of channel names occurring free in P and we write $\{y/x\}$ for the capture-avoiding substitution of y for the free occurrences of x. We impose a well-formedness condition on processes so that, in every sub-term of the form $\overline{x}(y, z)(P \mid Q)$, we have $y \notin \mathsf{fn}(Q)$ and $z \notin \mathsf{fn}(P)$.

We omit any concrete syntax for representing infinite processes. Instead, we work directly with infinite trees obtained by corecursively unfolding contractive equations of the form $A(x_1, \dots, x_n) = P$. For each such equation, we assume that $\mathsf{fn}(P) \subseteq \{x_1, \dots, x_n\}$ and we write $A\langle y_1, \dots, y_n\rangle$ for its unfolding $P\{y_i/x_i\}_{1 \le i \le n}$.

Notation 1. *To reduce clutter due to the systematic use of bound outputs, by convention we omit the continuation called y in Figure 6.1 when its name is chosen to coincide with that of the channel x on which y is sent/received. For example, with this notation we have $x(z).P = x(z, x).P$ and $\mathsf{in}_i\, \overline{x}.P = \mathsf{in}_i\, \overline{x}(x).P$ and $\mathsf{case}\, x\{P, Q\} = \mathsf{case}\, x(x)\{P, Q\}$.* ⌟

A welcome side effect of adopting Notation 1 is that it gives the illusion of working with a session calculus in which the same channel x may be used repeatedly for multiple input/output operations, while in fact x is a linear channel used for exchanging a single message along with a fresh continuation that turns out to have the same name. If one takes this notation as native syntax for a session calculus, its linear π-calculus encoding [Dardha et al., 2017] turns out to be precisely the $\pi\mathsf{LIN}$ term it denotes. Besides, the idea of rebinding the same name over and over is widespread in session-based functional languages [Gay and Vasconcelos, 2010, Padovani, 2017] as it provides a simple way of "updating the type" of a session endpoint after each use.

Example 6.2.1. *Below we model the interaction informally described in Example 2.1.1 between **buyer** and **seller** using the syntactic sugar defined in Notation 1:*

$$\begin{aligned} &(x)(\mathit{Buyer}\langle x\rangle \mid \mathit{Seller}\langle x, y\rangle) \\ \mathit{Buyer}(x) &= \mathsf{rec}\, \overline{x}.(\mathsf{add}\, \overline{x}.\mathit{Buyer}\langle x\rangle \oplus \mathsf{pay}\, \overline{x}.\overline{x}()) \\ \mathit{Seller}(x, y) &= \mathsf{corec}\, x.\mathsf{case}\, x\{\mathit{Seller}\langle x, y\rangle, x().\overline{y}()\} \end{aligned}$$

At each round of the interaction, the buyer decides whether to add an item to the shopping cart and repeat the same behavior (left branch of the

[SP-LINK]	$x \leftrightarrow y \preccurlyeq y \leftrightarrow x$	
[SP-COMM]	$(x)(P \mid Q) \preccurlyeq (x)(Q \mid P)$	
[SP-ASSOC]	$(x)(P \mid (y)(Q \mid R)) \preccurlyeq (y)((x)(P \mid Q) \mid R)$	if $x \in \mathsf{fn}(Q)$, $y \notin \mathsf{fn}(P)$, $x \notin \mathsf{fn}(R)$

Figure 6.2: Structural pre-congruence of $\pi\mathsf{LIN}$

choice) or to pay *the seller and terminate (right branch of the choice). The seller reacts dually and signals its termination by sending a unit on the channel y. As we will see in Section 6.3,* $\mathsf{rec}\,\overline{x}$ *and* $\mathsf{corec}\,x$ *identify the points within processes where (co)recursive types are unfolded.*

If we were to define Buyer using distinct bound names we would write an equation like

$$Buyer(x) = \mathsf{rec}\,\overline{x}(y).(\mathsf{add}\,\overline{y}(z).Buyer\langle z\rangle \oplus \mathsf{pay}\,\overline{y}(z).\overline{z}())$$

and similarly for Seller. ⌟

6.2.2 Operational Semantics

The operational semantics of the calculus is given in terms of the *structural precongruence* relation $\preccurlyeq$ and the *reduction relation* $\rightarrow$ defined in Figures 6.2 and 6.3. As usual, structural precongruence relates processes that are syntactically different but semantically equivalent. In particular, [SP-LINK] states that linking x with y is the same as linking y with x, whereas [SP-COMM] and [SP-ASSOC] state the expected commutativity and associativity laws for parallel composition. Concerning the latter, the side condition $x \in \mathsf{fn}(Q)$ makes sure that Q (the process brought closer to P when the relation is read from left to right) is indeed connected with P by means of the channel x. Note that [SP-ASSOC] only states the right-to-left associativity of parallel composition and that the left-to-right associativity law $(x)((y)(P \mid Q) \mid R) \preccurlyeq (y)(P \mid (x)(Q \mid R))$ is derivable when $x \in \mathsf{fn}(Q)$. The reduction relation is mostly unremarkable. Links are reduced with [RP-LINK] by effectively merging the linked channels. All the reductions that involve the interaction between processes except [RP-UNIT] create new continuations channels that connect the reducts. The rule [RP-CHOICE] models the non-deterministic choice between two behaviors. Finally, [RP-CUT] and [RP-STRUCT] close reductions by cuts and structural precongruence. In the following we write $\Rightarrow$ for the reflexive, transitive closure of $\rightarrow$ and we say that P is *stuck* if there is no Q such that $P \rightarrow Q$.

$$\textsc{rp-link}\ \frac{}{(x)(x \leftrightarrow y \mid P) \rightarrow P\{y/x\}} \qquad \textsc{rp-unit}\ \frac{}{(x)(\overline{x}() \mid x().P) \rightarrow P}$$

$$\textsc{rp-pair}\ \frac{}{(x)(\overline{x}(z,y)(P_1 \mid P_2) \mid x(z,y).Q) \rightarrow (z)(P_1 \mid (y)(P_2 \mid Q))}$$

$$\textsc{rp-sum}\ \frac{}{(x)(\mathsf{in}_i\,\overline{x}(y).P \mid \mathsf{case}\,x(y)\{P_1, P_2\}) \rightarrow (y)(P \mid P_i)}\ i \in \{1,2\}$$

$$\textsc{rp-rec}\ \frac{}{(x)(\mathsf{rec}\,\overline{x}(y).P \mid \mathsf{corec}\,x(y).Q) \rightarrow (y)(P \mid Q)} \qquad \textsc{rp-choice}\ \frac{}{P_1 \oplus P_2 \rightarrow P_i}\ i \in \{1,2\}$$

$$\textsc{rp-cut}\ \frac{P \rightarrow Q}{(x)(P \mid R) \rightarrow (x)(Q \mid R)} \qquad \textsc{rp-struct}\ \frac{P \preccurlyeq P' \qquad P' \rightarrow Q' \qquad Q' \preccurlyeq Q}{P \rightarrow Q}$$

Figure 6.3: Reduction of $\pi\mathsf{LIN}$

6.3 Type System

In this section we present the typing rules for $\pi\mathsf{LIN}$. As usual we introduce typing contexts to track the type of the names occurring free in a process. A *typing context* is a finite map from names to types written $x_1 : S_1, \dots, x_n : S_n$. We use Γ and Δ to range over contexts, we write $\mathsf{dom}(\Gamma)$ for the domain of Γ and Γ, Δ for the union of Γ and Δ when $\mathsf{dom}(\Gamma) \cap \mathsf{dom}(\Delta) = \emptyset$. Typing judgments have the form $P \vdash \Gamma$ and we say that P is *quasi typed* in Γ. For the time being we say "quasi typed" and not "well typed" because some infinite derivations using the rules in Section 6.3.1 are invalid. Well-typed processes are quasi-typed processes whose typing derivation satisfies some additional validity conditions that we detail in Section 6.3.2.

6.3.1 Typing Rules

We say that P is *quasi typed* in Γ if the judgment $P \vdash \Gamma$ is coinductively derivable using the rules shown in Figure 6.4 Rule [AX] states

$$\frac{}{x \leftrightarrow y \vdash x : \varphi_\alpha, y : \varphi_\beta^\perp}\ [\textsc{ax}] \qquad \frac{P \vdash \Gamma, x : S \qquad Q \vdash \Delta, x : S^\perp}{(x)(P \mid Q) \vdash \Gamma, \Delta}\ [\textsc{cut}]$$

$$\frac{}{\mathsf{case}\, x\{\} \vdash \Gamma, x : \top}\ [\top] \qquad \frac{P \vdash \Gamma}{x().P \vdash \Gamma, x : \bot}\ [\bot] \qquad \frac{}{\overline{x}() \vdash x : \mathbf{1}}\ [\mathbf{1}]$$

$$\frac{P \vdash \Gamma, y : S, z : T}{x(y,z).P \vdash \Gamma, x : S \parr T}\ [\parr] \qquad \frac{P \vdash \Gamma, y : S \qquad Q \vdash \Delta, z : T}{\overline{x}(y,z)(P \mid Q) \vdash \Gamma, \Delta, x : S \otimes T}\ [\otimes]$$

$$\frac{P \vdash \Gamma, y : S \qquad Q \vdash \Gamma, y : T}{\mathsf{case}\, x(y)\{P, Q\} \vdash \Gamma, x : S \,\&\, T}\ [\&] \qquad \frac{P \vdash \Gamma, y : S_i}{\mathsf{in}_i\, \overline{x}(y).P \vdash \Gamma, x : S_1 \oplus S_2}\ [\oplus]$$

$$\frac{P \vdash \Gamma, y : S\{\nu X.S/X\}}{\mathsf{corec}\, x(y).P \vdash \Gamma, x : \nu X.S}\ [\nu] \qquad \frac{P \vdash \Gamma, y : S\{\mu X.S/X\}}{\mathsf{rec}\, \overline{x}(y).P \vdash \Gamma, x : \mu X.S}\ [\mu]$$

$$\frac{P \vdash \Gamma \qquad Q \vdash \Gamma}{P \oplus Q \vdash \Gamma}\ [\textsc{choice}]$$

Figure 6.4: Typing rules for $\pi\mathsf{LIN}$

that a link $x \leftrightarrow y$ is quasi typed provided that x and y have dual types, but not necessarily dual addresses. Rule [CUT] states that a process composition $(x)(P \mid Q)$ is quasi typed provided that P and Q use the linear channel x in complementary ways, one according to some type S and the other according to the dual type $S^\perp$. Note that the context Γ, Δ in the conclusion of the rule is defined provided that Γ and Δ have disjoint domains. This condition entails that P and Q do not share any channel other than x ensuring that the interation between P and Q may proceed without deadlocks. Rule $[\top]$ deals with a process that receives an empty message from channel x. Since this cannot happen, we allow the process to be quasi typed in any context. Rules $[\mathbf{1}]$ and $[\bot]$ concern the exchange of units. The former rule states that $\overline{x}()$ is quasi typed in a context that contains a single association for the x channel with type $\mathbf{1}$, whereas the latter rule removes x from the context (hence from the set of usable channels), requiring the continuation process to be quasi typed in the remaining context. Rules $[\otimes]$ and $[\parr]$ concern the exchange of pairs. The former rule requires the two forked processes P and Q to be quasi typed in the respective contexts enriched with associations for the continuation channels y and z being created. The latter rule requires the continuation process to be quasi typed in a context enriched with the channels extracted from the received pair. Rules [$\oplus$] and [$\&$] deal with the exchange of disjoint sums in the expected way. Rules [μ] and [ν] deal with fixed point operators by unfolding the (co)recursive type of the channel x. As in $\mu\mathsf{MALL}^\infty$, the two rules have exactly the same structure despite the fact that the two fixed point operators being used are dual to each other. Clearly, the behavior of least and greatest fixed points must be distinguished by some other means, as we will see in Section 6.3.2 when discussing the validity of a typing derivation. Finally, [CHOICE] deals with non-deterministic choices by requiring that each branch of a choice must be quasi typed in exactly the same typing context as the conclusion.

Besides the structural constraints imposed by the typing rules, we implicitly require that the types in the range of all typing contexts have pairwise disjoint addresses. This condition ensures that it is possible to uniquely trace a communication protocol in a typing derivation: if we have two channels x and y associated with two types φ_α and ψ_β such that $\alpha \leq \beta$, then we know that y is a continuation resulting from a communication that started from x. In a sense, x and y represent different moments in the same conversation.

Remark 6.3.1. The typing rules in Figure 6.4 except [CHOICE] are in one-to-one correspondence with those of the $\mu\mathsf{MALL}^\infty$ proof system [Baelde et al., 2016, Doumane, 2017]. Concerning [CHOICE], it does not alter in any way the context of the process being typed. This implies that the type system is a conservative extension of $\mu\mathsf{MALL}^\infty$. That is, if $P \vdash x_1 : S_1, \dots, x_n : S_n$ is coinductively derivable using the rules in Figure 6.4, then $\vdash S_1, \dots, S_n$ is coinductively derivable in $\mu\mathsf{MALL}^\infty$. The converse is also true, although the proof term P is not uniquely determined.

Example 6.3.1 (Buyer - Seller)**.** *Let us show that the system described in Example 6.2.1 is quasi typed. To this aim, let* $\varphi \overset{\mathsf{def}}{=} \mu X.X \oplus \mathbf{1}$ *and* $\psi \overset{\mathsf{def}}{=} \nu X.X \mathbin{\&} \bot$ *respectively be the formulas describing the behavior of Buyer and Seller on the channel* x*. Note that* $\psi = \varphi^\perp$ *and let* a *be an arbitrary atomic address. We derive*

$$
\dfrac{
  \dfrac{
    \dfrac{
      \dfrac{\vdots}{\mathit{Buyer}\langle x\rangle \vdash x : \varphi_{ail}}
    }{
      \textsf{add}\,\overline{x}.\mathit{Buyer}\langle x\rangle \vdash x : (\varphi \oplus \mathbf{1})_{ai}
    }\,[\oplus]
    \qquad
    \dfrac{
      \dfrac{}{\overline{x}() \vdash x : \mathbf{1}}\,[\mathbf{1}]
    }{
      \textsf{pay}\,\overline{x}.\overline{x}() \vdash x : (\varphi \oplus \mathbf{1})_{ai}
    }\,[\oplus]
  }{
    \textsf{add}\,\overline{x}.\mathit{Buyer}\langle x\rangle \oplus \textsf{pay}\,\overline{x}.\overline{x}() \vdash x : (\varphi \oplus \mathbf{1})_{ai}
  }\,[\textsc{choice}]
}{
  \mathit{Buyer}\langle x\rangle \vdash x : \varphi_a
}\,[\mu]
$$

and also

$$\dfrac{\dfrac{\vdots}{\mathit{Seller}\langle x,y\rangle \vdash x : \psi_{a^\perp il}, y : \mathbf{1}} \qquad \dfrac{\dfrac{}{\overline{y}() \vdash y : \mathbf{1}}\,[\mathbf{1}]}{x().\overline{y}() \vdash x : \bot, y : \mathbf{1}}\,[\bot]}{\dfrac{\mathsf{case}\, x\{\mathit{Seller}\langle x,y\rangle, x().\overline{y}()\} \vdash x : (\psi \mathbin{\&} \bot)_{a^\perp i}, y : \mathbf{1}}{\mathit{Seller}\langle x,y\rangle \vdash x : \psi_{a^\perp}, y : \mathbf{1}}\,[\nu]}\,[\&]$$

showing that Buyer and Seller are quasi typed. Note that both derivations are infinite, but for dual reasons. In Buyer the infinite branch corresponds to the behavior in which Buyer chooses to add one more item to the shopping cart. This choice is made independently of the behavior of other processes in the system. In Seller, the infinite branch corresponds to the behavior in which Seller receives one more add *message from Buyer. By combining these derivations we obtain*

$$\dfrac{\dfrac{\vdots}{\mathit{Buyer}\langle x\rangle \vdash x : \varphi_a} \qquad \dfrac{\vdots}{\mathit{Seller}\langle x,y\rangle \vdash x : \psi_{a^\perp}, y : \mathbf{1}}}{(x)(\mathit{Buyer}\langle x\rangle \mid \mathit{Seller}\langle x,y\rangle) \vdash y : \mathbf{1}}\,[\textsc{cut}]$$

showing that the system as a whole is quasi typed. ⌟

6.3.2 From Quasi Typed to Well Typed Processes

𝄢 As we have anticipated, there exist infinite typing derivations that are unsound from a logical standpoint, because they allow us to prove **0** or the empty sequent. Hence, the typing rules presented in Figure 6.4 must be combined with additional *validity conditions.*

Example 6.3.2. *Consider the non-terminating process* $\Omega(x) = \Omega\langle x\rangle \oplus \Omega\langle x\rangle$. *We obtain the following infinite derivation showing that* $\Omega\langle x\rangle$ *is quasi typed.*

$$\dfrac{\dfrac{\vdots}{\Omega\langle x\rangle \vdash x : \mathbf{0}} \qquad \dfrac{\vdots}{\Omega\langle x\rangle \vdash x : \mathbf{0}}}{\Omega\langle x\rangle \vdash x : \mathbf{0}}\,[\textsc{choice}]$$

As illustrated by the next example, there exist non-terminating processes that are quasi typed also in logically sound contexts.

Example 6.3.3 (Compulsive Buyer)**.** *Consider the following variant of the Buyer process*

$$\mathit{Buyer}(x) = \mathsf{rec}\,\overline{x}.\mathsf{add}\,\overline{x}.\mathit{Buyer}\langle x\rangle$$

that models a "compulsive buyer", namely a buyer that adds infinitely many items to the shopping cart but never pays. Using $\varphi \stackrel{\mathsf{def}}{=} \mu X.X \oplus \mathbf{1}$ and an arbitrary atomic address a we can build the following infinite derivation

$$\dfrac{\dfrac{\dfrac{\vdots}{Buyer\langle x\rangle \vdash x : \varphi_{ail}}}{\mathsf{add}\,\overline{x}.Buyer\langle x\rangle \vdash x : (\varphi \oplus \mathbf{1})_{ai}}\,[\oplus]}{Buyer\langle x\rangle \vdash x : \varphi_{a}}\,[\mu]$$

showing that this process is quasi typed. By combining this derivation with the one for Seller in Example 6.3.1 we obtain a derivation establishing that $(x)(Buyer\langle x\rangle \mid Seller\langle x, y\rangle)$ is quasi typed in the context $y : \mathbf{1}$, although this composition cannot terminate. ⌟

To rule out unsound derivations like those in Examples 6.3.2 and 6.3.3 it is necessary to impose a validity condition on derivations [Baelde et al., 2016, Doumane, 2017]. Roughly speaking, $\mu\mathsf{MALL}^{\infty}$'s validity condition requires every infinite branch of a derivation to be supported by the continuous unfolding of a greatest fixed point. In order to formalize this condition, we start by defining *threads*, which are sequences of types describing sequential interactions at the type level.

Definition 6.3.1 (Thread)**.** A *thread* of S is a sequence of types $(S_i)_{i \in o}$ for some $o \in \omega + 1$ such that $S_0 = S$ and $S_i \rightsquigarrow S_{i+1}$ whenever $i + 1 \in o$.

Hereafter we use t to range over threads.

Example 6.3.4. *Consider $\varphi \stackrel{\mathsf{def}}{=} \mu X.X \oplus \mathbf{1}$ from Example 6.1.1 we have that $t \stackrel{\mathsf{def}}{=} (\varphi_a, (\varphi \oplus \mathbf{1})_{ai}, \varphi_{ail}, \dots)$ is an infinite thread of φ_a.*

A thread is *stationary* if it has an infinite suffix of equal types. The thread t from Example 6.3.4 is not stationary. Among all threads, we are interested in finding those in which a ν-formula is unfolded infinitely often. These threads, called ν-threads, are precisely defined thus:

Definition 6.3.2 (ν-thread)**.** Let $t = (S_i)_{i \in \omega}$ be an infinite thread, let $\overline{t}$ be the corresponding sequence $(\overline{S_i})_{i \in \omega}$ of formulas and let $\mathsf{inf}(t)$ be the set of elements of $\overline{t}$ that occur infinitely often in $\overline{t}$. We say that t is a *ν-thread* if $\min \mathsf{inf}(t)$ is defined and is a ν-formula.

Example 6.3.5. *Consider the infinite thread t from Example 6.3.4. We have $\mathsf{inf}(t) = \{\varphi, \varphi + \mathbf{1}\}$ and $\min \mathsf{inf}(t) = \varphi$, so t is* not *a ν-thread because φ is not a ν-formula.*

Example 6.3.6. *Consider the following formulas*

$$\varphi \overset{\mathsf{def}}{=} \nu X.\mu Y.X + Y \qquad\qquad \psi \overset{\mathsf{def}}{=} \mu Y.\varphi + Y$$

Observe that ψ is the "unfolding" of φ. Now

$$t_1 \overset{\mathsf{def}}{=} (\varphi_a, \psi_{ai}, (\varphi + \psi)_{aii}, \varphi_{aiil}, \dots)$$

is a thread of φ_a such that $\mathsf{inf}(t_1) = \{\varphi, \psi, \varphi+\psi\}$ and we have $\mathsf{min}\,\mathsf{inf}(t_1) = \varphi$ because $\varphi \preceq \psi$, so t_1 is a ν-thread. If, on the other hand, we consider the thread

$$t_2 \overset{\mathsf{def}}{=} (\varphi_a, \psi_{ai}, (\varphi + \psi)_{aii}, \psi_{aiir}, (\varphi + \psi)_{aiiri}, \dots)$$

such that $\mathsf{inf}(t_2) = \{\psi, \varphi + \psi\}$ we have $\mathsf{min}\,\mathsf{inf}(t_2) = \psi$ because $\psi \preceq \varphi + \psi$, so t_2 is not a ν-thread.

Intuitively, the $\preceq$-minimum formula among those that occur infinitely often in a thread is the outermost fixed point operator that is being unfolded infinitely often. It is possible to show that this minimum formula is always well defined [Doumane, 2017]. If such minimum formula is a greatest fixed point operator, then the thread is a ν-thread.

Now we proceed by identifying threads along branches of typing derivations. To this aim, we provide a precise definition of *branch*.

Definition 6.3.3 (Branch)**.** A *branch* of a typing derivation is a sequence $(P_i \vdash \Gamma_i)_{i\in o}$ of judgments for some $o \in \omega + 1$ such that $P_0 \vdash \Gamma_0$ occurs somewhere in the derivation and $P_{i+1} \vdash \Gamma_{i+1}$ is a premise of the rule application that derives $P_i \vdash \Gamma_i$ whenever $i + 1 \in o$.

An infinite branch is valid if supported by a ν-thread that originates somewhere therein.

Definition 6.3.4 (Valid Branch)**.** Let $\gamma = (P_i \vdash \Gamma_i)_{i\in\omega}$ be an infinite branch in a derivation. We say that γ is *valid* if there exists $j \in \omega$ such that $(S_k)_{k\geq j}$ is a non-stationary ν-thread and S_k is in the range of Γ_k for every $k \geq j$.

Example 6.3.7. *The infinite branch in the typing derivation for Seller of Example 6.1.1 is valid since it is supported by the ν-thread $(\psi_{a^\perp}, (\psi \mathbin{\&} \perp)_{a^\perp i}, \psi_{a^\perp il}, \dots)$ where $\psi \overset{\mathsf{def}}{=} \nu X.X \mathbin{\&} \perp$ happens to be the $\preceq$-minimum formula that is unfolded infinitely often.*

Example 6.3.8. *The infinite branch in the typing derivation for Buyer of Example 6.3.3 is invalid, because the only infinite thread in it is $(\varphi_a, (\varphi \oplus \mathbf{1})_{ai}, \varphi_{ail}, \dots)$ which is not a ν-thread.*

A $\mu\mathsf{MALL}^\infty$ derivation is valid if so is every infinite branch in it [Baelde et al., 2016, Doumane, 2017]. For the purpose of ensuring fair termination, this condition is too strong because some infinite branches in a typing derivation may correspond to unfair executions that, by definition, we neglect insofar its termination is concerned. For example, the infinite branch in the derivation for *Buyer* of Example 6.1.1 corresponds to an unfair run in which the buyer insists on adding items to the shopping cart, despite it periodically has a chance of paying the seller and terminate the interaction. That typing derivation for *Buyer* would be considered an invalid proof in $\mu\mathsf{MALL}^\infty$ because the infinite branch is not supported by a ν-thread (in fact, there is a μ-formula that is unfolded infinitely many times along that branch, as in Example 6.3.3).

It is generally difficult to understand if a branch corresponds to a fair or unfair run because the branch describes the evolution of an incomplete process whose behavior is affected by the interactions it has with processes found in other branches of the derivation. However, we can detect (some) unfair branches by looking at the non-deterministic choices they traverse, since choices are made autonomously by processes. To this aim, we introduce the notion of *rank* to estimate the least number of choices a process can possibly make during its lifetime.

Definition 6.3.5 (Rank)**.** Let r and s range over the elements of $\mathbb{N}^\infty \overset{\mathsf{def}}{=} \mathbb{N} \cup \{\infty\}$ equipped with the expected total order $\leq$ and operation $+$ such that $r + \infty = \infty + r = \infty$. The *rank* of a process P, written $|P|$, is the least element of $\mathbb{N}^\infty$ such that

$$\begin{array}{rlrl}
|x \leftrightarrow y| &= 0 & |\mathsf{rec}\,\overline{x}(y).P| &= |P| \\
|\mathsf{case}\,x\{\}| &= 0 & |\mathsf{corec}\,x(y).P| &= |P| \\
|\overline{x}()| &= 0 & |\mathsf{case}\,x(y)\{P,Q\}| &= \max\{|P|,|Q|\} \\
|x().P| &= |P| & |P \oplus Q| &= 1 + \min\{|P|,|Q|\} \\
|x(y,z).P| &= |P| & |(x)(P \mid Q)| &= |P| + |Q| \\
|\mathsf{in}_i\,\overline{x}(y).P| &= |P| & |\overline{x}(y,z)(P \mid Q)| &= |P| + |Q|
\end{array}$$

Roughly, the rank of terminated processes is 0, that of processes with a single continuation P coincides with the rank of P, and that of processes spawning two continuations P and Q is the sum of the ranks of P and Q. Then, the rank of a sum input with continuations P and Q is conservatively estimated as the maximum of the ranks of P and Q, since we do not know which one will be taken, whereas the rank of a choice with continuations P and Q is 1 plus the minimum of the ranks of P and Q.

Example 6.3.9. *Consider Buyer and Seller from Example 6.2.1 and Ω from Example 6.3.2. Then have $|\mathit{Buyer}\langle x\rangle| = 1$, $|\mathit{Seller}\langle x, y\rangle| = 0$ and $|\Omega\langle x\rangle| = \infty$.*

Note that $|P|$ only depends on the structure of P but not on the actual names occurring in P. As a consequence, when P is defined by means of a *finite* system of equations, the value of $|P|$ too can be determined by a *finite* system of equations.

Example 6.3.10. *Consider the definition of Buyer found in Example 6.2.1. In order to compute $|\mathit{Buyer}\langle x\rangle|$ we consider the system of equations*

$$\begin{array}{rcl}
|\mathit{Buyer}\langle\bullet\rangle| & = & |\mathsf{add}\,\overline{\bullet}(\bullet).\mathit{Buyer}\langle\bullet\rangle \oplus \mathsf{pay}\,\overline{\bullet}(\bullet).\overline{\bullet}()| \\
|\mathsf{add}\,\overline{\bullet}(\bullet).\mathit{Buyer}\langle\bullet\rangle \oplus \mathsf{pay}\,\overline{\bullet}(\bullet).\overline{\bullet}()| & = & 1 + \min\{ \\
 & & \qquad |\mathsf{add}\,\overline{\bullet}(\bullet).\mathit{Buyer}\langle\bullet\rangle|, \\
 & & \qquad |\mathsf{pay}\,\overline{\bullet}(\bullet).\overline{\bullet}()|\} \\
|\mathsf{add}\,\overline{\bullet}(\bullet).\mathit{Buyer}\langle\bullet\rangle| & = & |\mathit{Buyer}\langle\bullet\rangle| \\
|\mathsf{pay}\,\overline{\bullet}(\bullet).\overline{\bullet}()| & = & |\overline{\bullet}()| \\
|\overline{\bullet}()| & = & 0
\end{array}$$

where we have used a placeholder $\bullet$ in place of every channel name occurring in these terms. ⌟

Every such system of equations can be thought of as a function $\mathcal{F} : (\mathbb{N}^\infty)^n \to (\mathbb{N}^\infty)^n$ on the complete lattice $(\mathbb{N}^\infty)^n$ ordered by the pointwise extension of $\leq$ in $\mathbb{N}^\infty$. Note that $\mathcal{F}$ is monotone, because all the operators occurring in the definition of rank (see Definition 6.3.5) are monotone. So, $\mathcal{F}$ has a least fixed point by the Knaster-Tarski theorem and the rank of P is the component of this fixed point that corresponds to $|P|$.

Example 6.3.11. *For system of equations in Example 6.3.10 we have $n = 5$ and we have*

$$\mathcal{F}(x_1, x_2, x_3, x_4, x_5) = (x_2, 1 + \min\{x_3, x_4\}, x_1, x_5, 0)$$

whose least solution is $(1, 1, 1, 0, 0)$. Now $|\mathit{Buyer}\langle x\rangle|$ corresponds to the first component of this solution, that is 1. ⌟

Definition 6.3.6. A branch is *fair* if it traverses finitely many, finitely-ranked choices.

A finitely-ranked choice is at finite distance from a region of the process in which there are no more choices. An *unfair* branch gets close to such region infinitely often, but systematically avoids entering it. Note that every finite branch is also fair, but there are fair branches that are infinite.

Example 6.3.12. *All the infinite branches inside the derivation of Example 6.3.2 and the only infinite branch in the derivation for $Seller\langle x, y\rangle$ of Example 6.3.1 are fair since they do not traverse any finitely-ranked choice. On the contrary, the only infinite branch in the derivation for $Buyer\langle x\rangle$ of the Example 6.3.1 is unfair since it traverses infinitely many finitely-ranked choices. All fair branches in the same derivation for Buyer are finite.*

At last we can define our notion of well-typed process.

Definition 6.3.7 (Well-Typed Process)**.** We say that P is *well typed* in Γ, written $P \Vdash \Gamma$, if the judgment $P \vdash \Gamma$ is derivable and each fair, infinite branch in its derivation is valid.

Example 6.3.13. Ω *is ill typed since the fair, infinite branches in Example 6.3.2 are all invalid.*

Theorem 6.3.1 (Soundness)**.** *If $P \Vdash x : \mathbf{1}$ and $P \Rightarrow Q$ then $Q \Rightarrow \overline{x}()$.*

As for the session-based calculi (Chapters 4 and 5) Theorem 6.3.1 entails all the good properties we expect from well-typed processes: *failure freedom* (no unguarded sub-process $\mathsf{case}\, y\{\}$ ever appears), *deadlock freedom* (if the process stops it is terminated), *lock freedom* [Kobayashi, 2002a, Padovani, 2014] (every pending action can be completed in finite time) and *junk freedom* (every channel can be depleted).

6.3.3 Examples

We dedicate the rest of the section to show some more involved examples. In Example 6.3.14 we show the implementation of a *parallel programming pattern*. Example 6.3.15 models a simple *Fwd* that has the peculiarity of unfolds a leas fixed point infinitely many times. At last, in Example 6.3.16 we model a slot machine.

Example 6.3.14 (Parallel Programming)**.** *In this example we see a* $\pi\mathsf{LIN}$ *modeling of a* parallel programming pattern *whereby a Work process creates an unbounded number of workers each one dedicated to an independent task and a Gather process collects and combines the partial results from the workers. The processes Work and Gather are defined as follows:*

$$\begin{aligned} Work(x) &= \mathsf{rec}\, \overline{x}.(\mathsf{complex}\, \overline{x}.\overline{x}(y)(\overline{y}() \mid Work\langle x\rangle) \oplus \mathsf{simple}\, \overline{x}.\overline{x}()) \\ Gather(x, z) &= \mathsf{corec}\, x.\mathsf{case}\, x\{x(y).y().Gather\langle x, z\rangle, x().\overline{z}()\} \end{aligned}$$

At each iteration the Work process non-deterministically decides whether the task is complex *(left hand side of the choice) or* simple *(right hand side of the choice). In the first case, it bifurcates into a new worker, which in the example simply sends a unit on y, and another instance of itself. In the second case it terminates. The Gather process joins the results from all the workers before signalling its own termination by sending a unit on z. Note that the number of actions Gather has to perform before terminating is unbounded, as it depends on the non-deterministic choices made by Work.*

Below is a typing derivation for Work where $\varphi \overset{\text{def}}{=} \mu X.(\mathbf{1} \otimes X) \oplus \mathbf{1}$ and a is an arbitrary atomic address:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\overline{y}() \vdash y : \mathbf{1}}\,[\mathbf{1}] \qquad \dfrac{\vdots}{Work\langle x\rangle \vdash x : \varphi_{ailr}}}{\overline{x}(y)(\overline{y}() \mid Work\langle x\rangle) \vdash x : (\mathbf{1} \otimes \varphi)_{ail}}\,[\otimes]}{\textsf{complex}\,\overline{x} \ldots \vdash x : ((\mathbf{1} \otimes \varphi) \oplus \mathbf{1})_{ai}}\,[\oplus] \qquad \dfrac{}{\textsf{simple}\,\overline{x}.\overline{x}() \vdash x : \ldots}\,[\mathbf{1}], [\oplus]}{\dfrac{\textsf{complex}\,\overline{x}.\overline{x}(y)(\overline{y}() \mid Work\langle x\rangle) \oplus \textsf{simple}\,\overline{x}.\overline{x}() \vdash x : ((\mathbf{1} \otimes \varphi) \oplus \mathbf{1})_{ai}}{Work\langle x\rangle \vdash x : \varphi_{\alpha}}\,[\mu]}\,[\text{CHOICE}]$$

Note that the only infinite branch in this derivation is unfair because it traverses infinitely many choices with rank $1 = |Work\langle x\rangle|$. So, Work is well typed.

Concerning Gather, we obtain the following typing derivation where $\psi \overset{\text{def}}{=} \nu X.(\bot \parr X) \,\&\, \bot$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{Gather\langle x, z\rangle \vdash x : \psi_{a^\perp ilr}, z : \mathbf{1}}}{y().Gather\langle x, z\rangle \vdash x : \psi_{a^\perp ilr}, y : \bot, z : \mathbf{1}}\,[\bot]}{x(y).y().Gather\langle x, z\rangle \vdash x : (\bot \parr \psi)_{a^\perp il}, z : \mathbf{1}}\,[\parr] \qquad \dfrac{}{x().\overline{z}() \vdash x : \bot, z : \mathbf{1}}}{\dfrac{\textsf{case}\, x\{x(y).y().Gather\langle x, z\rangle, x().\overline{z}()\} \vdash x : ((\bot \parr \psi) \,\&\, \bot)_{a^\perp i}, z : \mathbf{1}}{Gather\langle x, z\rangle \vdash x : \psi_{a^\perp}, z : \mathbf{1}}\,[\nu]}\,[\&]$$

Here too there is just one infinite branch, which is fair and supported by the ν-thread $t = (\psi_{a^\perp}, ((\bot \parr \psi) \,\&\, \bot)_{a^\perp i}, (\bot \parr \psi)_{a^\perp il}, \psi_{a^\perp ilr}, \ldots)$. Indeed, all the formulas in $\overline{t}$ occur infinitely often and $\mathsf{min\,inf}(t) = \psi$ which is a ν-formula. Hence, Gather is well typed and so is the composition $(x)(Work\langle x\rangle \mid Gather\langle x, z\rangle)$ in the context $z : \mathbf{1}$. We conclude that the program is fairly terminating, despite the fact that the composition of Work and Gather may grow arbitrarily large because Work may spawn an unbounded number of workers. $\lrcorner$

Example 6.3.15 (Forwarder)**.** *In this example we illustrate a deterministic, well-typed process that unfolds a least fixed point infinitely many times. In particular, we consider once again the formulas* $\varphi \stackrel{\mathsf{def}}{=} \mu X.X \oplus \mathbf{1}$ *and* $\psi \stackrel{\mathsf{def}}{=} \nu X.X \mathbin{\&} \bot$ *and the process Fwd defined by the equation*

$$\mathit{Fwd}(x,y) = \mathsf{corec}\,x.\mathsf{rec}\,\overline{y}.\mathsf{case}\,x\{\mathsf{in}_1\,\overline{y}.\mathit{Fwd}\langle x,y\rangle, \mathsf{in}_2\,\overline{y}.x().\overline{y}()\}$$

which forwards the sequence of messages received from channel x to channel y. We derive

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\mathit{Fwd}\langle x,y\rangle \vdash x:\psi_{ail}, y:\varphi_{bil}}}{\mathsf{in}_1\,\overline{y}\ldots \vdash x:\psi_{ail}, y:(\varphi\oplus\mathbf{1})_{bi}}[\oplus] \qquad \dfrac{\dfrac{}{x().\overline{y}() \vdash x:\bot, y:\mathbf{1}}[\mathbf{1}],[\bot]}{\mathsf{in}_2\,\overline{y}\ldots \vdash x:\bot, y:(\varphi\oplus\mathbf{1})_{bi}}[\oplus]}{\mathsf{case}\,x\{\mathsf{in}_1\,\overline{y}.\mathit{Fwd}\langle x,y\rangle, \mathsf{in}_2\,\overline{y}.x().\overline{y}()\} \vdash x:(\psi \mathbin{\&} \bot)_{ai}, y:(\varphi\oplus\mathbf{1})_{bi}}[\&]}{\mathit{Fwd}\langle x,y\rangle \vdash x:\psi_a, y:\varphi_b}[\nu],[\mu]$$

and observe that $|\mathit{Fwd}\langle x,y\rangle| = 0$. *This typing derivation is valid because the only infinite branch is fair and supported by the* ν*-thread of* ψ_a*. Note that* $\varphi = \psi^\perp$ *and that the derivation proves an instance of* [AX]*. In general, the axiom is admissible in* $\mu\mathsf{MALL}^\infty$ *[Baelde et al., 2016].* ⌟

Example 6.3.16 (Slot Machine)**.** *Rank finiteness is not a necessary condition for well typedness. As an example, consider the system* $(x)(\mathit{Player}\langle x\rangle \mid \mathit{Machine}\langle x,y\rangle)$ *where*

$$\begin{aligned}
\mathit{Player}(x) &= \mathsf{rec}\,\overline{x}.(\mathsf{play}\,\overline{x}.\mathsf{case}\,x\{\mathit{Player}\langle x\rangle, \mathsf{rec}\,\overline{x}.\mathsf{quit}\,\overline{x}.\overline{x}()\} \oplus \mathsf{quit}\,\overline{x}.\overline{x}()) \\
\mathit{Machine}(x,y) &= \mathsf{corec}\,x. \\
&\quad\ \mathsf{case}\,x\{\mathsf{win}\,\overline{x}.\mathit{Machine}\langle x,y\rangle \oplus \mathsf{lose}\,\overline{x}.\mathit{Machine}\langle x,y\rangle, x().\overline{y}()\}
\end{aligned}$$

which models a game between a player and a slot machine. At each round, the player decides whether to play *or to* quit*. In the first case, the slot machine answers with either* win *or* lose*. If the player* wins*, it also* quits*. Otherwise, it repeats the same behavior. It is possible to show that* $\mathit{Player}\langle x\rangle \vdash x:\varphi_a$ *and* $\mathit{Machine}\langle x,y\rangle \vdash x:\psi_{a^\perp}, y:\mathbf{1}$ *are derivable where* $\varphi \stackrel{\mathsf{def}}{=} \mu X.(X \mathbin{\&} X)\oplus\mathbf{1}$ *and* $\psi \stackrel{\mathsf{def}}{=} \nu X.(X\oplus X)\mathbin{\&}\bot$*. For convenience, we define* $\varphi' \stackrel{\mathsf{def}}{=} (\varphi\mathbin{\&}\varphi)\oplus\mathbf{1}$

and we abbreviate Player to P. Note that φ' is the unfolding of φ.

$$\begin{prooftree}
\AxiomC{$\vdots$}
\UnaryInfC{$P\langle x\rangle \vdash x : \varphi_{aill}$}
\AxiomC{}
\RightLabel{$[\mathbf{1}]$}
\UnaryInfC{$\overline{x}() \vdash x : \mathbf{1}$}
\RightLabel{$[\oplus]$}
\UnaryInfC{$\mathsf{quit}\,\overline{x}.\overline{x}() \vdash x : \varphi'_{ailri}$}
\RightLabel{$[\mu]$}
\UnaryInfC{$\mathsf{rec}\,\overline{x}\ldots \vdash x : \varphi_{ailr}$}
\RightLabel{$[\&]$}
\BinaryInfC{$\mathsf{case}\,x\{P\langle x\rangle, \mathsf{rec}\,\overline{x}\ldots\} \vdash x : (\varphi \mathbin{\&} \varphi)_{ail}$}
\RightLabel{$[\oplus]$}
\UnaryInfC{$\mathsf{play}\,\overline{x}.\mathsf{case}\,x\{P\langle x\rangle, \mathsf{rec}\,\overline{x}\ldots\} \vdash x : \varphi'_{ai}$}
\AxiomC{}
\RightLabel{$[\mathbf{1}]$}
\UnaryInfC{$\overline{x}() \vdash x : \mathbf{1}$}
\RightLabel{$[\oplus]$}
\UnaryInfC{$\mathsf{quit}\,\overline{x}.\overline{x}() \vdash x : \varphi'_{ai}$}
\BinaryInfC{$\mathsf{play}\,\overline{x}\ldots \oplus \mathsf{quit}\,\overline{x}.\overline{x}() \vdash x : \varphi'_{ai}$}
\RightLabel{$[\mu]$}
\UnaryInfC{$P\langle x\rangle \vdash x : \varphi_a$}
\end{prooftree}$$

Note that the only infinite branch in the derivation for Player is unfair since $|Player\langle x\rangle| = 1$. Hence, Player is well typed.

Below is the typing derivation showing that Machine is quasi typed. For convenience, we define $\psi' \overset{\mathsf{def}}{=} \psi \oplus \psi$, we abbreviate Machine to M and we show the derivation only for the win *branch since the* lose *is the same.*

$$\begin{prooftree}
\AxiomC{$\vdots$}
\UnaryInfC{$M\langle x,y\rangle \vdash x : \psi_{bill}, y : \mathbf{1}$}
\RightLabel{$[\oplus]$}
\UnaryInfC{$\mathsf{win}\,\overline{x}\ldots \vdash x : \psi'_{bil}, y : \mathbf{1}$}
\AxiomC{$\vdots$}
\RightLabel{$[\oplus]$}
\UnaryInfC{$\mathsf{lose}\,\overline{x}\ldots \vdash \ldots$}
\RightLabel{$[\textsc{choice}]$}
\BinaryInfC{$\mathsf{win}\,\overline{x}\ldots \oplus \mathsf{lose}\,\overline{x}\ldots \vdash x : \psi'_{bil}, y : \mathbf{1}$}
\AxiomC{}
\RightLabel{$[\mathbf{1}]$}
\UnaryInfC{$\overline{y}() \vdash y : \mathbf{1}$}
\RightLabel{$[\bot]$}
\UnaryInfC{$x().\overline{y}() \vdash x : \bot, y : \mathbf{1}$}
\RightLabel{$[\&]$}
\BinaryInfC{$\mathsf{case}\,x\{\mathsf{win}\,\overline{x}\ldots \oplus \mathsf{lose}\,\overline{x}\ldots, x().\overline{y}()\} \vdash x : (\psi' \mathbin{\&} \bot)_{bi}, y : \mathbf{1}$}
\RightLabel{$[\nu]$}
\UnaryInfC{$M\langle x,y\rangle \vdash x : \psi_b, y : \mathbf{1}$}
\end{prooftree}$$

There are infinitely many branches in the derivation for Machine accounting for all the sequences of win *and* lose *choices that can be made. Since $|Machine\langle x,y\rangle| = \infty$, all these branches are fair but also valid. So, the system as a whole is well typed.* ⌟

6.4 Correctness

In this section we present the proof of Theorem 6.3.1. In general, the proof technique follows again that presented in Sections 4.3 and 5.3. However, the different scenario implies different design choices. Hence, we think that it is worth showing a more detailed sketch of the proof of Theorem 6.3.1 by introducing the main intermediate results and by relating them to $\mu\mathsf{MALL}^\infty$. Then, as in Sections 4.3 and 5.3, we present all the detailed proofs.

6.4.1 Proof Sketch

Here we informally explain the main results that are needed in order to prove Theorem 6.3.1. Roughly, the most important lemmas tell that typing is preserved by reduction (*subject reduction*) and that a well typed process can always reach termination (*weak termination*).

Theorem 6.4.1 (Subject Reduction)**.** *If $P \Vdash \Gamma$ and $P \to Q$ then $Q \Vdash \Gamma$.*

All reductions in Figure 6.3 except those for non-deterministic choices correspond to cut-elimination steps in a quasi typing derivation. As an illustration, below is a fragment of derivation tree for two processes exchanging a pair of y and z on channel x.

$$\dfrac{\dfrac{\dfrac{\vdots}{P \vdash \Gamma, y : S} \quad \dfrac{\vdots}{Q \vdash \Delta, z : T}}{\overline{x}(y,z)(P \mid Q) \vdash \Gamma, \Delta, x : S \otimes T}\,[\otimes] \qquad \dfrac{\dfrac{\vdots}{R \vdash \Gamma', y : S^{\perp}, z : T^{\perp}}}{x(y,z).R \vdash \Gamma', x : S^{\perp} \parr T^{\perp}}\,[\parr]}{(x)(\overline{x}(y,z)(P \mid Q) \mid x(y,z).R) \vdash \Gamma, \Delta, \Gamma'}\,[\textsc{cut}]$$

As the process reduces, the quasi typing derivation is rearranged so that the cut on x is replaced by two cuts on y and z. The resulting quasi typing derivation is shown below.

$$\dfrac{\dfrac{\vdots}{P \vdash \Gamma, y : S} \qquad \dfrac{\dfrac{\vdots}{Q \vdash \Delta, z : T} \quad \dfrac{\vdots}{R \vdash \Gamma', y : S^{\perp}, z : T^{\perp}}}{(z)(Q \mid R) \vdash \Delta, \Gamma', y : S^{\perp}}\,[\textsc{cut}]}{(y)(P \mid (z)(Q \mid R)) \vdash \Gamma, \Delta, \Gamma'}\,[\textsc{cut}]$$

It is also interesting to observe that, when $P \to Q$, the reduct Q is well typed in the same context as P but its rank may be different. In particular, the rank of Q can be *greater* than the rank of P. Recalling that the rank of a process estimates the number of choices that the process must perform to terminate, the fact that the rank of Q increases means that Q *moves away* from termination instead of getting closer to it (we will see an instance where this phenomenon occurs in Example 6.3.14). What really matters is that a well-typed process is weakly terminating. This is the second key property ensured by our type system.

Lemma 6.4.1 (Weak Termination)**.** If $P \Vdash x : \mathbf{1}$ then $P \Rightarrow \overline{x}()$.

The proof of Lemma 6.4.1 is a refinement of the cut elimination property of $\mu\mathsf{MALL}^\infty$. Essentially, the only new case we have to handle is when a choice $P_1 \oplus P_2$ "emerges" towards the bottom of the typing derivation, meaning that it is no longer guarded by any action. In this case, we reduce the choice to the P_i with smaller rank, which is guaranteed to lay on a fair branch of the derivation. An auxiliary result used in the proof of Lemma 6.4.1 is that our type system is a conservative extension of $\mu\mathsf{MALL}^\infty$.

Lemma 6.4.2 (Conservativity)**.** If $P \Vdash x_1 : S_1, \dots, x_n : S_n$ is derivable then $\vdash S_1, \dots, S_n$ is derivable in $\mu\mathsf{MALL}^\infty$.

Then, the proof of Theorem 6.3.1 is a simple consequence of Theorem 6.4.1 and Lemma 6.4.1. The combination of Theorems 2.2.1 and 6.3.1 also guarantees the termination of every fair run of the process.

Corollary 6.4.1 (Fair Termination)**.** *If $P \Vdash x : \mathbf{1}$ then P is fairly terminating.*

Observe that zero-ranked process do not contain any non-deterministic choice. In that case, every infinite branch in their typing derivation is fair and our validity condition coincides with that of $\mu\mathsf{MALL}^\infty$. As a consequence, we obtain the following strengthening of Corollary 6.4.1:

Proposition 6.4.1. *If $P \Vdash x : \mathbf{1}$ and $|P| = 0$ then P is terminating.*

For regular processes (those consisting of finitely many distinct sub-trees, up to renaming of bound names) it is possible to adapt the algorithm that decides the validity of a $\mu\mathsf{MALL}^\infty$ proof so that it decides the validity of a $\pi\mathsf{LIN}$ typing derivation.

6.4.2 Subject Reduction

We detail the proof of the *subject reduction* theorem (see Theorem 6.4.1). Notably, the presence of [RP-STRUCT] in the reduction rules implies a *subject congruence* lemma, stating that typing is preserved by structural pre-congruence. The same happened in Sections 4.3 and 5.3.

Lemma 6.4.3 (Substitution)**.** If $P \vdash \Gamma, x : S$ and $y \notin \mathsf{dom}(\Gamma)$, then $P\{y/x\} \vdash \Gamma, y : S$.

Proof. A simple application of the coinduction principle. $\square$

Lemma 6.4.4 (Quasi Subject Congruence)**.** If $P \vdash \Gamma$ and $P \preccurlyeq Q$, then $Q \vdash \Gamma$.

Proof. By induction on the derivation of $P \preccurlyeq Q$ and by cases on the last rule applied. We only discuss the base cases.

Case [SP-LINK]. Then $P = x \leftrightarrow y \preccurlyeq y \leftrightarrow x = Q$. From [AX] we deduce that there exist φ, α and β such that $\Gamma = x : \varphi_\alpha, y : \varphi_\beta^\perp$. We conclude with an application of [AX].

Case [SP-COMM]. Then $P = (x)(P_1 \mid P_2) \preccurlyeq (x)(P_2 \mid P_1) = Q$. From [CUT] we deduce that there exist Γ_1, Γ_2, and S such that

- $\Gamma = \Gamma_1, \Gamma_2$
- $P_1 \vdash \Gamma_1, x : S$
- $P_2 \vdash \Gamma_2, x : S^\perp$

We conclude with an application of [CUT].

Case [SP-ASSOC]. Then $P = (x)(P_1 \,|\, (y)(P_2 \,|\, P_3)) \preccurlyeq (y)((x)(P_1 \,|\, P_2) \,|\, P_3) = Q$ and $x \in \mathsf{fn}(P_2)$. From [CUT] we deduce that there exist Γ_1, Γ_2, Γ_3, S, T such that

- $\Gamma = \Gamma_1, \Gamma_2, \Gamma_3$
- $P_1 \vdash \Gamma_1, x : S$
- $P_2 \vdash \Gamma_2, x : S^\perp, y : T$
- $P_3 \vdash \Gamma_3, y : T^\perp$

We conclude with two applications of [CUT]. □

Lemma 6.4.5 (Quasi Subject Reduction)**.** If $P \to Q$ then $P \vdash \Gamma$ implies $Q \vdash \Gamma$.

Proof. By induction on $P \to Q$ and by cases on the last rule applied.

Case [RP-LINK]. Then $P = (x)(x \leftrightarrow y \mid R) \to R\{y/x\} = Q$. From [CUT] and [AX] we deduce that there exist φ_α, φ_β and Δ such that $\Gamma = y : \varphi_{\beta^\perp}^\perp, \Delta$ and $R \vdash \Delta, x : \varphi_{\beta^\perp}^\perp$ We conclude by applying lemma 6.4.3.

Case [RP-UNIT]. Then $P = (x)(\overline{x}() \,|\, x().Q) \to Q$. From [CUT], [$\mathbf{1}$] and [$\perp$] we conclude $Q \vdash \Gamma$.

Case [RP-PAIR]. Then $P = (x)(\overline{x}(y,z)(P_1 \mid P_2) \mid x(y,z).P_3) \to (y)(P_1 \mid (z)(P_2 \mid P_3)) = Q$. From [CUT], [$\otimes$] and [$\parr$] we deduce that there exist Γ_1, Γ_2, Γ_3, S and T such that

- $\Gamma = \Gamma_1, \Gamma_2, \Gamma_3$

- $P_1 \vdash \Gamma_1, y : S$
- $P_2 \vdash \Gamma_2, z : T$
- $P_3 \vdash \Gamma_3, y : S^\perp, z : T^\perp$

We conclude with two applications of [CUT].

Case [RP-SUM]. Then $P = (x)(\mathsf{in}_i\,\overline{x}(y).R \mid \mathsf{case}\,x(y)\{P_1, P_2\}) \rightarrow (y)(R \mid P_i) = Q$ for some $i \in \{1, 2\}$. From [CUT], [$\oplus$] and [&] we deduce that there exist Γ_1, Γ_2, S_1 and S_2 such that

- $\Gamma = \Gamma_1, \Delta$
- $R \vdash \Gamma_1, y : S_i$
- $P_1 \vdash \Gamma_2, y : S_1^\perp$
- $P_2 \vdash \Gamma_2, y : S_2^\perp$

We conclude with an application of [CUT].

Case [RP-REC]. Then $P = (x)(\mathsf{rec}\,\overline{x}(y).P_1 \,|\, \mathsf{corec}\,x(y).P_2) \rightarrow (y)(P_1 \,|\, P_2) = Q$. From [CUT], [$\mu$] and [$\nu$] we deduce that there exist Γ_1, Γ_2 and S such that

- $\Gamma = \Gamma_1, \Gamma_2$
- $P_1 \vdash \Gamma_1, y : S\{\mu X.S/X\}$
- $P_2 \vdash \Gamma_2, y : S^\perp\{\nu X.S^\perp/X\}$

We conclude with an application of [CUT].

Case [RP-CHOICE]. Then $P = P_1 \oplus P_2 \rightarrow P_i = Q$ for some $i \in \{1, 2\}$. From [CHOICE] we conclude that $P_i \vdash \Gamma$.

Case [RP-CUT]. Then $P = (x)(P' \mid R) \rightarrow (x)(Q' \mid R) = Q$ and $P' \rightarrow Q'$. From [CUT] we deduce that there exist Γ_1, Γ_2 and S such that

- $\Gamma = \Gamma_1, \Gamma_2$
- $P' \vdash \Gamma_1, x : S$
- $R \vdash \Gamma_2, x : S^\perp$

Using the induction hypothesis on $P' \rightarrow Q'$ we deduce $Q' \vdash \Gamma_1, x : S$. We conclude with an application of [CUT].

Case [RP-STRUCT]. Then $P \preccurlyeq P'$, $P' \rightarrow Q'$ and $Q' \preccurlyeq Q$ for some P', Q'. From Lemma 6.4.4 we deduce $P' \vdash \Gamma$. Using the induction hypothesis on $P' \rightarrow Q'$ we deduce $Q' \vdash \Gamma$. We conclude by applying Lemma 6.4.4 once more. □

Proof of Theorem 6.4.1. From the hypothesis $P \Vdash \Gamma$ we deduce $P \vdash \Gamma$. Using Lemma 6.4.5 we deduce $Q \vdash \Gamma$. Now, let γ be an infinite fair branch in the typing derivation for $Q \vdash \Gamma$. By inspecting the proof of Lemma 6.4.5 we observe that γ can be decomposed as $\gamma = \gamma_1\gamma_2$ where γ_2 is a branch in the typing derivation for $Q \vdash \Gamma$. From the fact that γ is fair we deduce that so is γ_2. From the hypothesis $P \Vdash \Gamma$ we deduce that γ_2 is valid, namely there is a ν-thread t in it. Then t is a ν-thread also of γ, that is γ is also valid. We conclude $Q \Vdash \Gamma$. $\square$

6.4.3 Proximity and Multicuts

𝄢 The cut elimination result of $\mu\mathsf{MALL}^\infty$ is proved by introducing a *multicut* rule that collapses several unguarded cuts in a single rule having a variable number of premises. The usefulness of working with multicuts is that they prevent infinite sequences of reductions where two cuts are continuously permuted in a non-productive way and no real progress is made in the cut elimination process. In the type system of $\pi\mathsf{LIN}$ we do not have a typing rule corresponding to the multicut. At the same time, the troublesome permutations of cut rules correspond to applications of the associativity law of parallel composition, namely of the [SP-ASSOC] pre-congruence rule. That is, the introduction of the multicut rule in the cut elimination proof of $\mu\mathsf{MALL}^\infty$ corresponds to working with $\pi\mathsf{LIN}$ terms considered equal up to structural pre-congruence. Nonetheless, since the $\pi\mathsf{LIN}$ reduction rules require two processes willing to interact on some channel x to be the children of the same application of [CUT], it is not entirely obvious that *not* introducing an explicit multicut rule allows us to perform in $\pi\mathsf{LIN}$ all the principal reductions that are performed during the cut elimination process in a $\mu\mathsf{MALL}^\infty$ proof.

Here we show that this is actually the case by proving a so-called *proximity lemma*, showing that every well-typed process can be rewritten in a structurally pre-congruent one where any two processes in which the same channel x occurs free are the children of a cut on x. To this aim, we introduce *process contexts* to refer to unguarded sub-terms of a process. Process contexts are processes with a single unguarded *hole* $[\,]$ and are generated by the following grammar:

$$\textbf{Process context} \qquad \mathcal{C}, \mathcal{D} \quad ::= \quad [\,] \quad | \quad (x)(\mathcal{C} \mid P) \quad | \quad (x)(P \mid \mathcal{C})$$

As usual we write $\mathcal{C}[P]$ for the process obtained by replacing the hole in $\mathcal{C}$ with P. Note that this substitution is not capture-avoiding in general and some free names occurring in P may be captured by binders in $\mathcal{C}$. We write $\mathsf{bn}(\cdot)$ for the function inductively defined by

$$\mathsf{bn}([\,]) = \emptyset \qquad \text{and} \qquad \mathsf{bn}((x)(\mathcal{C} \mid P)) = \mathsf{bn}((x)(P \mid \mathcal{C})) = \{x\} \cup \mathsf{bn}(\mathcal{C})$$

The next lemma shows that a cut on x can always be pushed down towards any process in which x occurs free.

Lemma 6.4.6. If $x \in \mathsf{fn}(P) \setminus \mathsf{bn}(\mathcal{C})$, then $(x)(\mathcal{C}[P] \mid Q) \preccurlyeq \mathcal{D}[(x)(P \mid Q)]$ for some $\mathcal{D}$.

Proof. By induction on the structure of $\mathcal{C}$ and by cases on its shape.

Case $\mathcal{C} = [\,]$. We conclude by taking $\mathcal{D} \overset{\mathsf{def}}{=} [\,]$ using the reflexivity of $\preccurlyeq$.

Case $\mathcal{C} = (y)(\mathcal{C}' \mid R)$. From the hypothesis $x \in \mathsf{fn}(P) \setminus \mathsf{bn}(\mathcal{C})$ we deduce $x \neq y$ and $x \in \mathsf{fn}(P) \setminus \mathsf{bn}(\mathcal{C}')$. Using the induction hypothesis we deduce that there exists $\mathcal{D}'$ such that $(x)(\mathcal{C}'[P]|Q) \preccurlyeq \mathcal{D}'[(x)(P|Q)]$. Take $\mathcal{D} \overset{\mathsf{def}}{=} (y)(\mathcal{D}'|R)$. We conclude

$$\begin{array}{rcll}
(x)(\mathcal{C}[P] \mid Q) & = & (x)((y)(\mathcal{C}'[P] \mid R) \mid Q) & \text{by definition of } \mathcal{C} \\
& \preccurlyeq & (x)(Q \mid (y)(\mathcal{C}'[P] \mid R)) & \text{by [SP-COMM]} \\
& \preccurlyeq & (y)((x)(Q \mid \mathcal{C}'[P]) \mid R) & \text{from } x \in \mathsf{fn}(P) \setminus \mathsf{bn}(\mathcal{C}) \\
& & & \text{and [SP-ASSOC]} \\
& \preccurlyeq & (y)((x)(\mathcal{C}'[P] \mid Q) \mid R) & \text{by [SP-COMM]} \\
& \preccurlyeq & (y)(\mathcal{D}'[(x)(P \mid Q)] \mid R) & \text{by induction hypothesis} \\
& = & \mathcal{D}[(x)(P \mid Q)] & \text{by definition of } \mathcal{D}
\end{array}$$

Case $\mathcal{C} = (y)(R \mid \mathcal{C}')$. Analogous to the previous case. □

The proximity lemma is then proved by applying Lemma 6.4.6 twice.

Lemma 6.4.7 (Proximity)**.**
If $x \in \mathsf{fn}(P_i) \setminus \mathsf{bn}(\mathcal{C}_i)$ for $i = 1, 2$, then $\mathcal{C}[(x)(\mathcal{C}_1[P_1] \mid \mathcal{C}_2[P_2]) \preccurlyeq \mathcal{D}[(x)(P_1 \mid P_2)]$ for some $\mathcal{D}$.

Proof. It suffices to apply Lemma 6.4.6 twice, thus:

$$\begin{array}{rll}
\mathcal{C}[(x)(\mathcal{C}_1[P_1] \mid \mathcal{C}_2[P_2]) & \preccurlyeq \mathcal{C}[\mathcal{D}_1[(x)(P_1 \mid \mathcal{C}_2[P_2])]] & \text{using the hypothesis} \\
& & \text{and Lemma 6.4.6} \\
& \preccurlyeq \mathcal{C}[\mathcal{D}_1[(x)(\mathcal{C}_2[P_2] \mid P_1)]] & \text{by [SP-ASSOC]} \\
& \preccurlyeq \mathcal{C}[\mathcal{D}_1[\mathcal{D}_2[(x)(P_2 \mid P_1)]]] & \text{using the hypothesis} \\
& & \text{and Lemma 6.4.6} \\
& \preccurlyeq \mathcal{C}[\mathcal{D}_1[\mathcal{D}_2[(x)(P_1 \mid P_2)]]] & \text{by [SP-ASSOC]}
\end{array}$$

and we conclude by taking $\mathcal{D} \overset{\mathsf{def}}{=} \mathcal{C}[\mathcal{D}_1[\mathcal{D}_2]]$. □

Notably, Lemma 6.4.7 corresponds to Lemmas 4.3.9 and 5.3.10 that we proved in the session based scenarios.

6.4.4 Additional Properties

Most of the soundness proof of the type system relies on the cut elimination result of $\mu\mathsf{MALL}^\infty$. Here we gather some auxiliary definitions and properties of well-typed processes. First of all, we define a function that makes a "fair choice" among two processes P and Q, by selecting the one with smaller rank. If P and Q happen to have the same rank, we choose P by convention.

Definition 6.4.1. The *fair choice* among P and Q, written $\mathsf{fc}(P, Q)$, is defined by

$$\mathsf{fc}(P, Q) \stackrel{\mathsf{def}}{=} \begin{cases} P & \text{if } |P| \leq |Q| \\ Q & \text{otherwise} \end{cases}$$

From its definition we have $|\mathsf{fc}(P, Q)| = \min\{|P|, |Q|\}$. Next, we show that in a well-typed process there cannot be an infinite sequence of choices if we follow the fair ones. To this aim, we define a total function on processes that computes the length of the longest chain of subsequent fair choices.

Definition 6.4.2. Let $\mathsf{depth}(\cdot)$ be the function from processes to $\mathbb{N}^\infty$ such that

$$\mathsf{depth}(P) = \begin{cases} 1 + \mathsf{depth}(\mathsf{fc}(P_1, P_2)) & \text{if } P = P_1 \oplus P_2 \\ 0 & \text{otherwise} \end{cases}$$

Note that $\mathsf{depth}(\mathsf{fc}(P, Q)) < 1 + \mathsf{depth}(\mathsf{fc}(P, Q)) = \mathsf{depth}(P \oplus Q)$. We prove that, for well-typed processes, $\mathsf{depth}(\cdot)$ always yields a natural number. That is, in a well-typed process there is no infinite chain of fair choices.

Lemma 6.4.8. If $P \Vdash \Gamma$ then $\mathsf{depth}(P) \in \mathbb{N}$.

Proof. Suppose that $\mathsf{depth}(P) = \infty$. Then the derivation for $P \vdash \Gamma$ has an infinite branch $\gamma = (P_i \vdash \Gamma)_{i \in \omega}$ solely consisting of choices such that $|P_{i+1}| < |P_i|$ for every $i \in \omega$. Therefore, $|P_i| = \infty$ for every $i \in \omega$ and γ is fair. From the hypothesis that P is well typed we deduce that γ is also valid (see Definition 6.3.4), namely it has a non-stationary ν-thread. This contradicts the fact that the contexts in γ are all equal to Γ. We conclude that $\mathsf{depth}(P) \in \mathbb{N}$. □

Now we define a function $\lfloor \cdot \rfloor$ on well-typed processes to statically and fairly resolve all the choices of a process.

Definition 6.4.3. Let $\lfloor \cdot \rfloor$ be the function on well-typed processes such that $\lfloor P \oplus Q \rfloor = \lfloor \mathsf{fc}(P, Q) \rfloor$ and extended homomorphically to all the other process forms.

The fact that $\lfloor P \rfloor$ is uniquely defined when P is well typed is a consequence of Lemma 6.4.8. Indeed, any branch of P that contains infinitely many choices also contains infinitely many forms other than choices. Note that the range of $\lfloor \cdot \rfloor$ only contains zero-ranked processes.

The next two results prove that $\lfloor P \rfloor$ is well typed if so is P.

Lemma 6.4.9. If $P \Vdash \Gamma$ then $\lfloor P \rfloor \vdash \Gamma$.

Proof. We apply the coinduction principle to show that every judgment in the set

$$\mathcal{R} \overset{\mathsf{def}}{=} \{\lfloor P \rfloor \vdash \Gamma \mid P \Vdash \Gamma\}$$

is the conclusion of a rule in Figure 6.4 whose premises are also in $\mathcal{R}$. Let $Q \vdash \Gamma \in \mathcal{R}$. Then $Q = \lfloor P \rfloor$ for some P such that $P \Vdash \Gamma$. From Lemma 6.4.8 we deduce that $\mathsf{depth}(P) \in \mathbb{N}$. We reason by induction on $\mathsf{depth}(P)$ and by cases on the shape of P to show that $Q \vdash \Gamma$ is the conclusion of a rule in Figure 6.4 whose premises are also in $\mathcal{R}$. We only discuss a few cases, the others being similar or simpler.

Case $P = (x)(P_1 \mid P_2)$. Then $Q = \lfloor P \rfloor = (x)(\lfloor P_1 \rfloor \mid \lfloor P_2 \rfloor)$. From [CUT] we deduce that there exists Γ_1, Γ_2, S_1 and S_2 such that $P_i \Vdash \Gamma_i, x : S_i$ for $i = 1, 2$ and $\Gamma = \Gamma_1, \Gamma_2$ and $S_1 = S_2^{\perp}$. Then $\lfloor P_i \rfloor \vdash \Gamma_i, x : S_i \in \mathcal{R}$ by definition of $\mathcal{R}$ and we conclude observing that $Q \vdash \Gamma \in \mathcal{R}$ is the conclusion of [CUT].

Case $P = !\{P_1\}P_2$. Then $Q = \lfloor P \rfloor = \lfloor \mathsf{fc}(P_1, P_2) \rfloor$. From [CHOICE] we deduce $\mathsf{fc}(P_1, P_2) \Vdash \Gamma$. Since $\mathsf{depth}(\mathsf{fc}(P_1, P_2)) < \mathsf{depth}(P) \in \mathbb{N}$, we conclude using the induction hypothesis. □

Lemma 6.4.10. If $P \Vdash \Gamma$ then $\lfloor P \rfloor \Vdash \Gamma$.

Proof. Using Lemma 6.4.9 we deduce $\lfloor P \rfloor \vdash \Gamma$. Now, consider an infinite branch γ in the derivation for $\lfloor P \rfloor \vdash \Gamma$ and observe that γ is necessarily fair, since it does not traverse any choice. The branch γ corresponds to another infinite branch γ' in the derivation for $P \vdash \Gamma$ which always makes fair choices whenever it traverses a process of the form $P_1 \oplus P_2$. The only differences between γ and γ' are the judgments for the choices in P that have been resolved. Nonetheless, all of the contexts that occur in γ' also occur in γ because [CHOICE] does not affect typing contexts. If γ' traverses infinitely many choices, then γ' traverses infinitely many processes with strictly decreasing ranks, which must all be ∞. Therefore, γ' is fair. Since P is well typed we know that γ' is valid. Then, the ν-thread that witnesses the validity of γ' corresponds to a ν-thread that witnesses the validity of γ. We conclude that $\lfloor P \rfloor$ is well typed. □

Proof of Lemma 6.4.2. By Lemma 6.4.10 we deduce $\lfloor P \rfloor \Vdash x_1 : S_1, \dots, x_n : S_n$. Since $|\lfloor P \rfloor| = 0$, there are no applications of [CHOICE] in the derivation for $\lfloor P \rfloor \vdash x_1 : S_1, \dots, x_n : S_n$. That is, this derivation corresponds to a $\mu\mathsf{MALL}^\infty$ derivation and every infinite branch in it is fair. Since $\lfloor P \rfloor$ is well typed, we deduce that every infinite branch in this derivation is valid. That is, $\vdash S_1, \dots, S_n$ is derivable in $\mu\mathsf{MALL}^\infty$. □

The key property of zero-ranked processes that are well typed in a context $x : \mathbf{1}$ is that they are weakly terminating and they eventually reduce to $\overline{x}()$.

Lemma 6.4.11. If $P \Vdash x : \mathbf{1}$ and $|P| = 0$ then $P \Rightarrow \mathsf{close}\, x$.

Proof. Without loss of generality we may assume that P does not contain links. Indeed, the axiom [AX] is admissible in $\mu\mathsf{MALL}^\infty$ [Baelde et al., 2016, Proposition 10], so we may assume that links have been expanded into equivalent (choice-free) processes. We just note that, if the statement holds for link-free processes, then it holds also in the case P does contain links, the only difference being that the sequence of reductions that are needed to fully eliminate all the cuts resulting from an expanded link are replaced by a single occurrence of [RP-LINK].

From the hypothesis that P has rank 0 we deduce that P does not contain non-determimnistic choices and the derivation of $P \vdash x : \mathbf{1}$ does not contain applications of the [CHOICE] rule. So, every infinite branch of this derivation is fair. From the hypothesis that every infinite fair branch of this derivation is valid we deduce that every infinite branch of this derivation is valid. Then this derivation corresponds to a $\mu\mathsf{MALL}^\infty$ proof of $\vdash \mathbf{1}$. Observe that every principal reduction rule of $\mu\mathsf{MALL}^\infty$ [Doumane, 2017, Figure 3.2] corresponds to a reduction rule of $\pi\mathsf{LIN}$ (see Figure 6.3). Also, Lemma 6.4.7 guarantees that, whenever there are two processes in which the same channel name x occurs free we are always able to rewrite the process using structural precongruence so that the two sub-processes are the children of a cut on x. Finally, the cut elimination result for $\mu\mathsf{MALL}^\infty$ is proved by reducing bottom-most cuts, meaning that principal reductions (also called *internal reductions* [Doumane, 2017]) are only applied at the bottom of a $\mu\mathsf{MALL}^\infty$ proof. Therefore, each step in which a principal reduction is applied in the cut elimination result of $\mu\mathsf{MALL}^\infty$ can be mimicked by a reduction of $\pi\mathsf{LIN}$. From the cut elimination result for $\mu\mathsf{MALL}^\infty$ [Doumane, 2017, Proposition 3.5] we deduce that there exists no (fair) infinite sequence of principal reductions. That is, there exists a stuck Q such that $P \Rightarrow Q$. From Lemma 6.4.5 we deduce $Q \vdash x : \mathbf{1}$. Since the only cut-free $\mu\mathsf{MALL}^\infty$

proof of $\vdash \mathbf{1}$ consists of a single application of [$\mathbf{1}$], and since this proof corresponds to the proof that Q is quasi typed, we conclude that $Q = \mathsf{close}\, x$. $\square$

6.4.5 Soundness

𝄢 The last auxiliary result for proving Lemma 6.4.1 is to show that P weakly simulates $\lfloor P \rfloor$, namely that every reduction of a process $\lfloor P \rfloor$ can be mimicked by one or more reductions of P.

Lemma 6.4.12. If $P \Vdash \Gamma$ and $\lfloor P \rfloor \to Q$ then $P \Rightarrow R$ for some R such that $Q = \lfloor R \rfloor$.

Proof. From Lemma 6.4.8 we deduce $\mathsf{depth}(P) \in \mathbb{N}$. We proceed by double induction on $\mathsf{depth}(P)$ and on the derivation of $\lfloor P \rfloor \to Q$ and by cases on the last rule applied in the derivation of $P \vdash \Gamma$. Most cases are straightforward since the structure of $\lfloor P \rfloor$ and that of P only differ for non-deterministic choices, so we only discuss the case [CHOICE] in which $P = !\{P_1\}P_2$. Then $\mathsf{fc}(P_1, P_2) \vdash \Gamma$ and $\lfloor P \rfloor = \lfloor \mathsf{fc}(P_1, P_2) \rfloor \to Q$. Recall that $\mathsf{depth}(\mathsf{fc}(P_1, P_2)) < \mathsf{depth}(P) \in \mathbb{N}$. Using the induction hypothesis we deduce $\mathsf{fc}(P_1, P_2) \Rightarrow R$ for some R such that $Q = \lfloor R \rfloor$. We conclude by observing that $P \to \mathsf{fc}(P_1, P_2) \Rightarrow R$. $\square$

Lemma 6.4.12 can be easily generalized to arbitrary sequences of reductions.

Lemma 6.4.13. If $P \Vdash \Gamma$ and $\lfloor P \rfloor \Rightarrow Q$ then $P \Rightarrow R$ for some R such that $Q = \lfloor R \rfloor$.

Proof. A simple induction on the number of reductions in $\lfloor P \rfloor \Rightarrow Q$ using Lemma 6.4.12. $\square$

Proof of Lemma 6.4.1. Using Lemma 6.4.10 we deduce $\lfloor P \rfloor \Vdash x : \mathbf{1}$. Using Lemma 6.4.11 we deduce $\lfloor P \rfloor \Rightarrow \mathsf{close}\, x$. Using Lemma 6.4.13 and Lemma 6.4.5 we conclude $P \Rightarrow \mathsf{close}\, x$. $\square$

Finally, we can prove Theorem 6.3.1 and corollary 6.4.1.

Proof of Theorem 6.3.1. By induction on the number of reductions in $P \Rightarrow Q$, from the hypothesis $P \Vdash x : \mathbf{1}$ and Theorem 6.4.1 we deduce $Q \Vdash x : \mathbf{1}$. We conclude using Lemma 6.4.1. $\square$

Proof of Corollary 6.4.1.
Immediate consequence of Theorems 2.2.1 and 6.3.1. $\square$

6.5 Comparing the Type Systems

This last section concludes Part II so we dedicate it to a comparison between the two approaches to fair termination that we presented, that is, the session based one (see Chapters 4 and 5) and that about linear logic (in the present chapter). The most important result is that the two approaches *cannot* be actually compared since there exist examples that are accepted in the former and rejected in the other and viceversa. Here we provide two examples of this fact (see Examples 6.5.1 and 6.5.2). Then, such result leaves an open problem, that is establishing the meaning of fair subtyping in the linear logic scenario. Indeed, the type system in Section 6.3.1 does not use subtyping. What makes this research line interesting is how *infinite* behaviors are dealt with in the two approaches; the former relies on fair subtyping and the second on validity conditions. To conclude, in Example 6.5.3 we encode Example 4.2.11 in the linear logic context and we see how it is rejected by the type system in Section 6.3.1.

Example 6.5.1 (Forwarder)**.** *Here we recall the forwarder that we proved to be well typed in $\pi\mathsf{LIN}$ in Example 6.3.15. We model the same process using binary sessions and we try to prove that it is well typed using the type system in Chapter 4.*

$$Fwd(x,y) = x?\{\mathsf{ok}.y!\mathsf{ok}.Fwd\langle x,y\rangle, \mathsf{stop}.y!\mathsf{stop}.\mathsf{wait}\,x.\mathsf{close}\,y\}$$

where x and y are used according to S_x and S_y defined below

$$S_x = ?\{\mathsf{ok}.S_x, \mathsf{stop}.?\mathsf{end}\} \qquad S_y = !\{\mathsf{ok}.S_y, \mathsf{stop}.!\mathsf{end}\}$$

It is clear that, if we try to provide a typing derivation for Fwd, then we have to apply rule [TB-TAG] *that we report here for the sake of clarity.*

$$\frac{\forall i \in I : \Gamma, x : S_i \vdash^n P_i}{\Gamma, x : \pi\{\mathsf{m}_i : S_i\}_{i\in I} \vdash^n x\pi\{\mathsf{m}_i : P_i\}_{i\in I}}$$

In order to apply such rule the branches must be typed in some contexts that differ only with respect to x. Hence, we cannot apply [TB-TAG] *by taking into account S_x and S_y because the behavior of y changes as well. We can try to solve the problem by placing casts in the right positions*

$$Fwd(x,y) = x?\{\mathsf{ok}.\lceil y\rceil y!\mathsf{ok}.Fwd\langle x,y\rangle, \mathsf{stop}.\lceil y\rceil y!\mathsf{stop}.\mathsf{wait}\,x.\mathsf{close}\,y\}$$

Indeed y is used according to $!\mathsf{ok}.S_y$ and $!\mathsf{stop}.!\mathsf{end}$ in the left and right branches, respectively. Moreover, the two applications of fair subtyping are *correct and we can easily derive* $S_y \leqslant_1 !\mathsf{ok}.S_y$ *and* $S_y \leqslant_1 !\mathsf{stop}.!\mathsf{end}$. *Note that the weights are* 1 *because the types differ only on the topmost output, which means that the contravariant rule* [FSB-TAG-OUT-2] *for output must be applied. Now we can try to provide a typing derivation (we omit the rule names).*

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{x : S_x, y : S_y \vdash^n \mathit{Fwd}\langle x,y\rangle}}{x : S_x, y : S_y \vdash^n y!\mathsf{ok}.\mathit{Fwd}\langle x,y\rangle}}{x : S_x, y : !\mathsf{ok}.S_y \vdash^{n+1} \lceil y\rceil y!\mathsf{ok}.\mathit{Fwd}\langle x,y\rangle} \quad \dfrac{\dfrac{\dfrac{\dfrac{}{y : !\mathsf{end} \vdash^n \mathsf{close}\, y}}{x : ?\mathsf{end} : y : !\mathsf{end} \vdash^n \mathsf{wait}\, x.\mathsf{close}\, y}}{x : ?\mathsf{end}, y : !\mathsf{stop}.!\mathsf{end} \vdash^n !\mathsf{stop}\ldots}}{x : ?\mathsf{end}, y : S_y \vdash^{n+1} \lceil y\rceil \ldots}}{x : S_x, y : S_y \vdash^{n+1} \mathit{Fwd}\langle x,y\rangle}$$

where the right branch ends with an application of [TB-END] *which assigns rank* n. *Indeed,* [TB-TAG] *requires that the rank is an upper bound to the rank of the branches so we assign it in order to match that of the left one. Note that it is not possible to assign a finite rank to the process because it requires* $n = n + 1$, *hence the process if* cast unbounded *(Section 4.2.1).* ⌟

Example 6.5.2 (A Hopeful Player)**.** *The slot machine was a recurring example in Part II (see Examples 5.2.3 and 6.3.16). In Example 3.2.2 we described the behaviors of a fair and an unfair machine. The second one simply never lets the player win. Let* S, T *be the types from Example 3.2.2 such that*

$$\begin{array}{rcl} S & = & ?\mathsf{play}.(!\mathsf{win}.S + !\mathsf{lose}.S) + ?\mathsf{quit}.!\mathsf{end} \\ T & = & ?\mathsf{play}.!\mathsf{lose}.T + ?\mathsf{quit}.!\mathsf{end} \end{array}$$

Now we can model the two implementations of the machines. For the sake of simplicity, we adapt the one in Example 5.2.3 to the binary case. We dub SlotF *the fair machine and* SlotU *the unfair one.*

$$\begin{array}{rcl} \mathit{SlotF}(x) & \triangleq & x?\{\mathsf{play}.x!\{\mathsf{win}.\mathit{SlotF}\langle x\rangle, \mathsf{lose}.\mathit{SlotF}\langle x\rangle\}, \mathsf{quit}.\mathsf{close}\, x\} \\ \mathit{SlotU}(x) & \triangleq & x?\{\mathsf{play}.x!\mathsf{lose}.\mathit{SlotU}\langle x\rangle, \mathsf{quit}.\mathsf{close}\, x\} \end{array}$$

It is easy to see that $x : S \vdash^0 \mathit{SlotF}\langle x\rangle$ *and that* $x : S \not\vdash \mathit{SlotU}\langle x\rangle$ *since the* win *branch is missing. We can try to rewrite* SlotU *in order to be well-typed with* $x : S$. *First, we can place a cast in the* lose *branch.*

$$\mathit{SlotU}(x) \triangleq x?\{\mathsf{play}.\lceil x\rceil x!\mathsf{lose}.\mathit{SlotU}\langle x\rangle, \mathsf{quit}.\mathsf{close}\, x\}$$

where the cast is correct since we can derive $(!\mathsf{win}.S + !\mathsf{lose}.S) \leqslant_1 !\mathsf{lose}.S$. *The* 1 *weight is due to the fact that the two types differ only for the topmost* *output, which requires an application of* [FSB-TAG-OUT-2]. *Let us try to derive a typing judgment.*

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{x : S \vdash^n \mathit{SlotU}\langle x\rangle}}{x : !\mathsf{lose}.S \vdash^n x!\mathsf{lose}.\mathit{SlotU}\langle x\rangle}}{x : !\mathsf{win}.S + !\mathsf{lose}.S \vdash^{n+1} \lceil x \rceil x!\mathsf{lose}.\mathit{SlotU}\langle x\rangle} \qquad \dfrac{}{x : !\mathsf{end} \vdash^{n+1} \mathsf{close}\, x}}{x : S \vdash^{n+1} \mathit{SlotU}\langle x\rangle}$$

Hence, we cannot find a finite rank; the process is cast unbounded *(see Section 4.2.1). Alternatively we can try to place a cast before the first invocation of the process since it is clear that SlotU uses x according to T. Such cast would be invalid since in Example 3.2.2 we proved that $S \not\leqslant T$. Hence, the only typing judgment that we can derive is $x : T \vdash^0 \mathit{SlotU}\langle x\rangle$.*

*Now we can apply the same reasoning from the π*LIN *point of view. Let us model the fair (Example 6.3.16) and the unfair machines. We dub MF the fair machine and MU the unfair one.*

$$\begin{array}{lcl} MF(x,y) & \triangleq & \mathsf{corec}\, x. \\ & & \mathsf{case}\, x\{\mathsf{win}\, \overline{x}.MF\langle x,y\rangle \oplus \mathsf{lose}\, \overline{x}.MF\langle x,y\rangle, x().\overline{y}()\} \\ MU(x,y) & \triangleq & \mathsf{corec}\, x.\mathsf{case}\, x\{\mathsf{lose}\, \overline{x}.MU\langle x,y\rangle, x().\overline{y}()\} \end{array}$$

In Example 6.3.16 we proved that $MF\langle x,y\rangle \vdash x : \psi_b, y : \mathbf{1}$ and that MF is well-typed where $\psi \overset{\mathsf{def}}{=} \nu X.(X \oplus X) \,\&\, \bot$. At the same time we can prove that $MU\langle x,y\rangle \vdash x : \psi_b, y : \mathbf{1}$

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{MU\langle x,y\rangle \vdash x : \psi_{bilr}, y : \mathbf{1}}}{\mathsf{lose}\, \overline{x}.MU\langle x,y\rangle \vdash x : \psi'_{bil}, y : \mathbf{1}}\,[\oplus] \qquad \dfrac{\dfrac{}{\overline{y}() \vdash y : \mathbf{1}}\,[\mathbf{1}]}{x().\overline{y}() \vdash x : \bot, y : \mathbf{1}}\,[\bot]}{\mathsf{case}\, x\{\mathsf{lose}\, \overline{x}.MU\langle x,y\rangle, x().\overline{y}()\} \vdash x : (\psi' \,\&\, \bot)_{bi}, y : \mathbf{1}}\,[\&]}{MU\langle x,y\rangle \vdash x : \psi_b, y : \mathbf{1}}\,[\nu]$$

Moreover the process is well-typed since no choices are met. This example points out that we can type two different processes with the same type while in the session scenario we the correspondence between types and processes is tighter.

Now we can focus on the player side and we can try to model a hopeful player that plays until it receives win. *In that case, it stops playing. We first* *write it in the binary session scenario and then we encode it in πLIN.*

$$Player(x) \triangleq x!\mathsf{play}.x?\{\mathsf{win}.\mathsf{close}\,x, \mathsf{lose}.Player\langle x\rangle\}$$

where x is used according to $T_p = !\mathsf{play}.?\{\mathsf{win}.!\mathsf{end}, \mathsf{lose}.T_p\}$. It can be derived that $x : T_p \vdash^0 Player\langle x\rangle$. Note that the system consisting of MF and Player is fairly terminating. Indeed $S \sim T_p$.

Now we model the same process in πLIN. In particular, we slightly modify the player in Example 6.3.16.

$$P(x) = \mathsf{rec}\,\overline{x}.\mathsf{play}\,\overline{x}.\mathsf{case}\,x\{P\langle x\rangle, \mathsf{rec}\,\overline{x}.\mathsf{quit}\,\overline{x}.\overline{x}()\}$$

Let us look at the typing derivation with $\varphi \overset{\mathsf{def}}{=} \mu X.(X \,\&\, X) \oplus \mathbf{1}$.

$$\dfrac{\dfrac{\dfrac{\vdots}{P\langle x\rangle \vdash x : \varphi_{aill}} \qquad \dfrac{\dfrac{\dfrac{}{\overline{x}() \vdash x : \mathbf{1}}\,[\mathbf{1}]}{\mathsf{quit}\,\overline{x}.\overline{x}() \vdash x : \varphi'_{ailri}}\,[\oplus]}{\mathsf{rec}\,\overline{x}.\mathsf{quit}\,\overline{x}.\overline{x}() \vdash x : \varphi_{ailr}}\,[\mu]}{\dfrac{\mathsf{case}\,x\{P\langle x\rangle, \mathsf{rec}\,\overline{x}.\,\overline{\mathsf{quit}}x.\overline{x}()\} \vdash x : (\varphi \,\&\, \varphi)_{ail}}{\mathsf{play}\,\overline{x}.\mathsf{case}\,x\{P\langle x\rangle, \mathsf{rec}\,\overline{x}.\mathsf{quit}\,\overline{x}.\overline{x}()\} \vdash x : \varphi'_{ai}}\,[\oplus]}\,[\&]}{P\langle x\rangle \vdash x : \varphi_a}\,[\mu]$$

Note that now $|P\langle x\rangle| = 0$ since there are no choices. Hence, the infinite branch is fair because it traverses no choices. We can conclude that the $P\langle x\rangle$ is not well typed because because the infinite, fair branch is not valid. We recall that the player in Example 6.3.16 was well typed because the infinite branch was unfair as it contained infinitely many finitely ranked choices. ⌟

Examples 6.5.1 and 6.5.2 show that the type systems in Chapters 4 and 5 cannot be compared to that in Chapter 6. Example 6.5.1 illustrates a process that can be only typed in πLIN while Example 6.5.2 show a process that can be typed only in the session based calculi. Moreover Example 6.5.2 shows a peculiar difference between the type systems. In Chapters 4 and 5 the tight correspondence between types and processes allows us to use a *fair* type to prove that a *fair* process is well-typed. Such correspondence does not hold in πLIN as we proved that both the fair and the unfair slot machines can be typed with the same formula. We conclude this section showing how to encode delegation in πLIN and trying to encode Example 4.2.11 to see whether it is accepted or rejected in πLIN.

Definition 6.5.1 (Delegation in πLIN)**.** We show delegation how can be encoded in πLIN. Lindley and Morris [2015] provide an encoding from GV (a session typed functional language) to CP (a process calculus based on classical linear logic). Hence, they encode (binary) session types to the types of CP, that is, linear logic formulas. Concerning delegation, the encoding is provided as

$$\lfloor !T.S \rfloor \;=\; \overline{\lfloor T \rfloor} \otimes \lfloor S \rfloor \qquad \lfloor ?T.S \rfloor \;=\; \lfloor T \rfloor \parr \lfloor S \rfloor$$

(see [Lindley and Morris, 2015, Fig.9]). Following the encoding of types we extend the encoding to the processes in Section 4.1.

$$\lfloor x!(y).P \rfloor \;=\; \overline{x}(z)(y \leftrightarrow z \mid \lfloor P \rfloor) \qquad \lfloor x?(y) \rfloor \;=\; x(y)\lfloor P \rfloor$$

Example 6.5.3 (Delegation)**.** *In this example we encode Example 4.2.11 in πLIN and we see if we obtain a well-typed process or not. To encode delegation we refer to Definition 6.5.1. Let us first recall the process and the main types.*

$$\begin{aligned} & (y)((x)(A\langle x,y\rangle \mid B\langle x\rangle) \mid B\langle y\rangle) \\ A(x,y) &\triangleq x!\mathsf{more}.x!y.B\langle x\rangle \\ B(x) &\triangleq x?\{\mathsf{more} : x?(y).A\langle y,x\rangle, \mathsf{stop} : \mathsf{wait}\, x.\mathsf{done}\} \end{aligned}$$

where x and y are used according to S_A and T_A in $A(x,y)$ and x is used according to S_B in $B(x)$.

$$\begin{aligned} S_A &= !\mathsf{more}.!T_A.S_B \\ S_B &= ?\mathsf{more}.?S_A.T_A + ?\mathsf{stop}.?\mathsf{end} \\ T_A &= !\mathsf{more}.!T_A.S_B + !\mathsf{stop}.!\mathsf{end} \end{aligned}$$

Now we can look at the encoding of processes.

$$\begin{aligned} \lfloor A(x,y) \rfloor &= \mathsf{rec}\,\overline{x}.\mathsf{more}\,\overline{x}.\overline{x}(z)(y \leftrightarrow z \mid \lfloor B(x) \rfloor) \\ \lfloor B(x) \rfloor &= \mathsf{corec}\,x.\mathsf{case}\,x\{x(y)\lfloor A(y,x)\rfloor, x()\} \end{aligned}$$

The encoding of types is more involved. Since we do not have subtyping in πLIN, we can encode S_b and T_A. Moreover, if we replace $?S_A.T_A$ with

$?T_A.T_A$ in S_B, we obtain that $\overline{S_B} = T_A$. Let $\lfloor T_A \rfloor = \varphi_A$ and $\lfloor S_B \rfloor = \varphi_B$.

$$\begin{aligned}
\varphi_A = {} & \mu Y.(\varphi_A^\perp \otimes \varphi_B) \oplus \mathbf{1} \\
& \hfill \textit{by definition of } \lfloor T_A \rfloor \\
= {} & \mu Y.(\varphi_A^\perp \otimes ((Y \parr Y) \mathbin{\&} \bot)) \oplus \mathbf{1} \\
& \hfill \textit{by replacing } \varphi_B \\
= {} & \mu Y.((\nu X.(Y \parr ((X \otimes X) \oplus \mathbf{1})) \mathbin{\&} \bot) \otimes ((Y \parr Y) \mathbin{\&} \bot)) \oplus \mathbf{1} \\
& \hfill \textit{by replacing } \varphi_A^\perp \\
\varphi_B = {} & \nu X.(\varphi_A \parr \varphi_A) \mathbin{\&} \bot \\
& \hfill \textit{by definition of } \lfloor S_B \rfloor \\
= {} & \nu X.(\varphi_A \parr ((X \otimes X) \oplus \mathbf{1})) \mathbin{\&} \bot \\
& \hfill \textit{by replacing } \varphi_A \\
= {} & \nu X.((\mu Y.(X \otimes ((Y \parr Y) \mathbin{\&} \bot)) \oplus \mathbf{1}) \parr ((X \otimes X) \oplus \mathbf{1})) \mathbin{\&} \bot \\
& \hfill \textit{by replacing } \varphi_A
\end{aligned}$$

From these definition it can be noted that it the encoding is not straightforward since we do not know how/when actually unfold the formulas. Indeed we are not able to find the right types to prove that the process is quasi-typed.

Furthermore, assuming that we could find such types, it would turn out that B is well-typed while A is not as its infinite thread in not a ν one. ⌟

6.6 Related Work

On account of the known encodings of sessions into the linear π-calculus [Kobayashi, 2002b, Dardha et al., 2017, Scalas et al., 2017], πLIN belongs to the family of process calculi providing logical foundations to sessions and session types. Some representatives of this family are πDILL [Caires et al., 2016] and its variant equipped with a circular proof theory [Derakhshan and Pfenning, 2019, Derakhshan, 2021], CP [Wadler, 2014] and μCP [Lindley and Morris, 2016], among others. There are two main aspects distinguishing πLIN from these calculi. The first one is that these calculi take sessions as a native feature. This fact can be appreciated both at the level of processes, where session endpoints are *linearized* resources that can be used *multiple times* albeit in a sequential way, and also at the level of types, where the interpretation of the $\varphi \otimes \psi$ and $\varphi \parr \psi$ formulas is skewed so as to distinguish the type φ of the message being sent/received on a channel from the type ψ of the channel after the exchange has taken place. In contrast, πLIN adopts a more fundamental communication model based on *linear* channels, and is thus closer to the spirit of the encoding of linear logic proofs into the π-calculus proposed by Bellin and Scott [1994] while retaining the same expressiveness of the aforementioned calculi. To some extent, $\pi\mathsf{LIN}$ is also more general than those calculi, since a formula $\varphi \otimes \psi$ may be interpreted as a protocol that bifurcates into two independent sub-protocols φ and ψ (we have seen an instance of this pattern in Example 6.3.14). So, $\pi\mathsf{LIN}$ is natively equipped with the capability of modeling some multiparty interactions, in addition to all of the binary ones. A session-based communication model identical to $\pi\mathsf{LIN}$, but whose type system is based on intuitionistic rather than classical linear logic, has been presented by DeYoung et al. [2012]. In that work, the authors advocate the use of explicit continuations with the purpose of modeling an asynchronous communication semantics and they prove the equivalence between such model and a buffered session semantics. However, they do not draw a connection between the proposed calculus and the linear π-calculus [Kobayashi et al., 1999] through the encoding of binary sessions [Kobayashi, 2002b, Dardha et al., 2017] and, in the type system, the multiplicative connectives are still interpreted asymmetrically. The second aspect that distinguishes $\pi\mathsf{LIN}$ from the other calculi is its type system, which is still deeply rooted into linear logic and yet it ensures fair termination instead of progress [Caires et al., 2016, Wadler, 2014, Qian et al., 2021], termination [Lindley and Morris, 2016] or strong progress [Derakhshan and Pfenning, 2019, Derakhshan, 2021]. Fair termination entails progress, strong progress and lock freedom [Kobayashi, 2002a, Padovani, 2014], but at the same time it does not always rule out processes admitting infinite executions. Simply, infinite executions are deemed unrealistic because they are unfair.

Another difference between $\pi\mathsf{LIN}$ and other calculi based on linear logic is that its operational semantics is completely ordinary, in the sense that it does not include commuting conversions, reductions under prefixes, or the swapping of communication prefixes. The cut elimination result of $\mu\mathsf{MALL}^\infty$, on which the proof of Theorem 6.3.1 is based, works by reducing cuts from the bottom of the derivation instead of from the top [Baelde et al., 2016, Doumane, 2017, Baelde et al., 2022]. As a consequence, it is not necessary to reduce cuts guarded by prefixes or to push cuts deep into the derivation tree to enable key reductions in $\pi\mathsf{LIN}$ processes.

The extension of calculi based on linear logic with non-deterministic features has recently received quite a lot of attention. Rocha and Caires [2021] have proposed a session calculus with shared cells and non-deterministic choices that can model mutable state. Their typing rule for non-deterministic choices is the same as our own, but in their calculus choices do not reduce. Rather, they keep track of every possible evolution of a process to be able to prove a confluence result. Qian et al. [2021] introduce *coexponentials*, a new pair of modalities that enable the modeling of concurrent clients that compete in order to interact with a shared server that processes requests sequentially. In this setting, non-determinism arises from the unpredictable order in which different clients are served. Interestingly, the coexponentials are derived by resorting to their semantics in terms of least and greatest fixed points. For this reason, the cut elimination result of $\mu\mathsf{MALL}^{\infty}$ might be useful to attack the termination proof in their setting.

Part III

Agda Mechanizations *(Allegro Moderato)*

Chapter 7

A Library For (Generalized) Inference Systems

In this chapter we present a library for supporting generalized inference systems (see Section 1.2) in Agda [The Agda Team, 2022]. Summarizing from Section 1.2, an *inference system* [Aczel, 1977, Leroy and Grall, 2009, Sangiorgi, 2011], that is, a set of (meta-)rules stating that a consequence can be derived from a set of premises, is a simple, general and widely-used way to express and reason about a recursive definition. In most cases such recursive definition is seen as inductive, that is, the denoted set consists of the elements with a finite derivation. This enables *inductive reasoning*, that is, to prove that the elements an inductively defined set satisfy a property, it is enough to show that, for each (meta-)rule, the property holds for the consequence assuming that it holds for the premises. In other cases, the recursive definition is seen as coinductive, that is, the denoted set consists of the elements with a possibly infinite derivation. This enables *coinductive reasoning*, that is, to prove that all the elements satisfying a property belong to the coinductively defined set, it is enough to show that, when the property holds for an element, it can be derived from premises for which the property holds as well. Recently, a generalization of inference systems has been proposed [Ancona et al., 2017b, Dagnino, 2019, 2021] which handles cases where neither the inductive, nor the purely coinductive intepretation provides the desired meaning. This approach is called *flexible coinduction*, and, correspondingly, coinductive reasoning is generalized as well by a principle which is called *bounded coinduction.*

The Agda proof assistant [The Agda Team, 2022] offers language constructs to inductively/coinductively define predicates, and correspondingly built-in proof principles. However, in this way the recursive definition is monolithic, and hard-wired with its chosen interpretation.

Remark 7.0.1. In Ciccone [2020] we deeply investigated the built-in features of Agda by inspecting all the different ways a (co)inductive predicate can be defined. We found out that, while pure (co)inductive definitions are easily supported, a predicate mixing both approaches led to complex codes with many duplicated notions due to the lack of modularity (see Remark 7.1.3).

Our aim, instead, is to provide an Agda library allowing the user to express a recursive definition as an *instance of a parametric type* of inference systems. In this way, the user is not committed from the beginning to a given interpretation but, rather, gets for free a bunch of properties which have been proved once and for all, including the inductive and coinductive intepretation and the corresponding proof principles. Moreover, it is possible to define composition operators on inference systems, for instance union and restriction. Finally, flexible coinduction is modularly obtained as well, by composing in a certain way the interpretations of two inference systems.

Indexed containers [Altenkirch et al., 2015] provide a way to specify possibly recursive definitions of predicates independently from their interpretation and are supported in the Agda standard library. An Agda implementation of inference systems can be provided by seeing them as indexed containers. However, this approach requires to structure definitions in an unusual way. Indeed, inference systems are usually presented through a (finite) set of *meta-rules*, denoting all the rules which can be obtained by instantiating meta-variables with values satisfying the side condition. Hence, we provide a different implementation following this schema, to allow users to write their own inference system in an Agda format which closely resembles that "on paper". We then prove that the two implementations are equivalent, showing that every indexed container can be encoded in terms of meta-rules and viceversa.

Remark 7.0.2 (Agda Version & Reference). It is important to point out that the material presented in Chapters 7 and 8 holds as long as Agda is consistent. During the development of the notions we are going to present, Agda received many updates. As an example, an inconsistency in mixing *sized types* and *coinductive records* has been found. The library is available on GitHub (see Ciccone et al. [2021b]) and has been tested with the Agda 2.6.2.2. In case of future Agda updates, we will try to keep the material updated. ⌟

The chapter is structured as follows. In Section 7.1 we show the mechanization of the meta theory of (generalized) inference systems (see Section 1.2). In particular, we present the main datatypes to encode inference systems in Agda, how to obtain their interpretations and, most important, the proof principles. We provide the formalization of the examples that we introduce in Section 1.2 since they give an overview of all the features of the library. Then, in Section 7.2 we show a more involved example, that is, a lambda calculus with divergence and we prove soundness and completeness of its big-step semantics. At last, in Section 7.3 we investigate the relation between inference systems and indexed containers [Altenkirch et al., 2015] that are supported by the standard library.

7.1 Meta Theory

In this section we describe the main definitions of the library. Concerning the meta-theory, the reader can refer to Section 1.2. Notably, we first present an approach mimicking meta-rules. Then we introduce the notions of *interpretations* and we prove the *principles.* At last, we formalize the examples we presented in Section 1.2.

7.1.1 (Meta-)Rules/Inference System

As anticipated, the aim of the Agda library is to allow a user to write meta-rules as "on paper". To illustrate this format, let us consider, e.g., the *allPos* example from Section 1.2:

$$\frac{\mathit{allPos}\ \mathit{xs}}{\mathit{allPos}\ x{:}\mathit{xs}}\ x > 0$$

In a meta-rule, we have *meta-variables*, which range over certain sets, in a way possibly restricted by a *side condition.* We call *context* the set of the instantiations of meta-variables which satisfy the side-condition, hence produce a rule of the inference system. In the example, there are two meta-variables, x and xs, which range over $\mathbb{N}$ and $\mathbb{N}^\infty$, respectively, with the restriction that x should be positive. Hence the context is $\{\langle x, l\rangle \in \mathbb{N} \times \mathbb{N}^\infty \mid x > 0\}$, see Section 7.1.3 for the Agda version of this meta-rule. Correspondingly, the Agda declaration in Figure 7.1 defines a meta-rule as a record, parametric on the universe `U`. The first two components are the context and a set of positions for premises. For each element of the context

```
record MetaRule {ℓc ℓp : Level} (U : Set ℓu) : Set _ where
  field
    Ctx : Set ℓc
    Pos : Set ℓp
    prems : Ctx → Pos → U
    conclu : Ctx → U

  RF[_] : ∀{ℓ} → (U → Set ℓ) → (U → Set _)
  RF[_] P u =
        Σ[ c ∈ Ctx ] (u ≡ conclu c × (∀ p → P (prems c p)))

  RClosed : ∀{ℓ} → (U → Set ℓ) → Set _
  RClosed P = ∀ c → (∀ p → P (prems c p)) → P (conclu c)

record IS {ℓc ℓp ℓn : Level} (U : Set ℓu) : Set _ where
  field
    Names : Set ℓn
    rules : Names → MetaRule {ℓc} {ℓp} U

  ISF[_] : ∀{ℓ} → (U → Set ℓ) → (U → Set _)
  ISF[_] P u = Σ[ rn ∈ Names ] RF[ rules rn ] P u

  ISClosed : ∀{ℓ} → (U → Set ℓ) → Set _
  ISClosed P = ∀ rn → RClosed (rules rn) P
```

Figure 7.1: MetaRule datatype

(instantiation of meta-variables satisfying the side condition), the last two components produce the premises, one for each position, and the conclusion of the rule obtained by this instantiation.

Recall that in Agda the declaration U : Set introduces the type (set) U, and P : U → Set the dependent type (predicate on U) P. For each element u of U, P u is the type of the proofs that u satifies P, hence P u inhabited means that u satisfies P. To avoid paradoxes, not every Agda type is in Set; there is an infinite sequence Set 0, Set 1, . . . , Set ℓ, . . . such that Set ℓ : Set (suc ℓ), where ℓ is a *level*, and Set is an abbreviation for Set 0. The programmer can write a wildcard for a level which can be inferred; to make the Agda code reported in the paper lighter, we sometimes use a wildcard even for a level which is explicit in the real code.

Remark 7.1.1. In the Agda code in this section, predicates P : U → Set encode subsets of the universe, so we speak of subsets and membership, rather than of predicates and satisfaction, to closely follow the previous

```
record FinMetaRule {ℓc n} (U : Set ℓu) : Set _ where
  field
    Ctx : Set ℓc
    comp : Ctx → Vec U n × U

  from : MetaRule {ℓc} {zero} U
  from .MetaRule.Ctx = Ctx
  from .MetaRule.Pos = Fin n
  from .MetaRule.prems c i = get (proj₁ (comp c)) i
  from .MetaRule.conclu c = proj₂ (comp c)
```

Figure 7.2: Finitary meta-rule

formulation. ⌟

The function RF[_] encodes the inference operator associated with the meta-rule. Given a subset P of the universe, u belongs to the resulting subset if we can find an instantiation c of meta-variables satisfying the side condition, producing u as conclusion, and, for each position, a premise in P. Note the use of existential quantification Σ[x ∈ A] B where B depends on x. The predicate RClosed encodes the property of being closed with respect to the meta-rule. A subset P of the universe is closed if, for each instantiation c of the meta-variables satisfying the side-condition, if all the premises are in P then the conclusion is in P as well. Note the use of universal quantification ∀ (x : A) → B, where B depends on x. Finally, an inference system is defined in Figure 7.1 as a record, parametric on the universe U, consisting of a set of meta-rule names and a family of meta-rules. The function ISF [_] and the predicate ISClosed are defined composing those given for a single meta-rule.

Since in practical cases metarules are very often *finitary*, that is, premises are a finite set, the library also offers an interface to write a (finitary) meta-rule (see Figure 7.2), by providing, besides the context, two components which are the *vector* of premises, with fixed length n, and the conclusion. The injection from transforms this more concrete format in the generic one for meta-rules, by specifying that the set of positions is Fin n (the set of indexes from 0 to $n-1$).

```
data Ind⟦_⟧ {ℓc ℓp ℓn : Level}
(is : IS {ℓc} {ℓp} {ℓn} U) : U → Set _ where
  fold : ∀ {u} → ISF[ is ] Ind⟦ is ⟧ u → Ind⟦ is ⟧ u

record CoInd⟦_⟧ {ℓc ℓp ℓn : Level}
 (is : IS {ℓc} {ℓp} {ℓn} U) (u : U) : Set _ where
  coinductive
  constructor cofold_
  field
    unfold : ISF[ is ] CoInd⟦ is ⟧ u

data SCoInd⟦_⟧ {ℓc ℓp ℓn : Level}
(is : IS {ℓc} {ℓp} {ℓn} U) : U → Size → Set _ where
  sfold : ∀ {u i} → ISF[ is ] (λ u → Thunk (SCoInd⟦ is ⟧ u) i) u
      → SCoInd⟦ is ⟧ u i
```

Figure 7.3: (Co)inductive interpretations - datatype

7.1.2 Interpretations and Principles

𝄢 Recall that the inductive interpretation $\mathsf{Ind}[\![\mathcal{I}]\!]$ of an inference system $\mathcal{I}$ is the set of elements of the universe which have a finite proof tree, and finite proof trees are, in turn, inductively defined, that is, by a least fixed point operator. In Agda, inductive structures are encoded as *datatypes* (see Figure 7.3), which specify their constructors. For each `u`, `Ind ⟦ is ⟧ u` is the type of the proofs that `u` satisfies `Ind ⟦ is ⟧`, which are essentially the finite proof trees [1] for `u`. Indeed, the `fold` constructor, given a proof that `u` can be derived by applying a rule from premises belonging to `Ind ⟦ is ⟧`, which essentially consists of a rule with conclusion `u` and finite proof trees for its premises, builds a finite proof tree for `u`.

The coinductive interpretation $\mathsf{CoInd}[\![\mathcal{I}]\!]$ (see Figure 7.3), instead, is the set of elements of the universe which have a possibly infinite proof tree, and possibly infinite proof trees are, in turn, coinductively defined, that is, by a greatest fixed point operator. For each `u`, `CoInd ⟦ is ⟧ u` is the type of the proofs that `u` satisfies `CoInd ⟦ is ⟧`, which are essentially the possibly infinite proof trees for `u`, and analogously for `SCoInd ⟦ is ⟧`.

Remark 7.1.2. In Agda, coinductive structures can be encoded in two different ways: either as *coinductive records* [Abel et al., 2013], or as datatypes by using the mechanism of *thunks* (suspended computations) together with

[1] With some more structure, since the Agda proofs keep trace of the applied meta-rules.

```
_⊓_ : ∀ {ℓc ℓp ℓn ℓ}{U : Set ℓu} → IS {ℓc} {ℓp} {ℓn} U
  → (U → Set ℓ) → IS {ℓc ⊔ ℓ} {_} {_} U
(is ⊓ P) .Names = is .Names
(is ⊓ P) .rules rn = addSideCond (is .rules rn) P

_∪_ : ∀{ℓc ℓp ℓn ℓn'}{U : Set ℓ} → IS {ℓc} {ℓp} {ℓn} U
  → IS {_} {_} {ℓn'} U → IS {_} {_} {ℓn ⊔ ℓn'} U
(is1 ∪ is2) .Names = (is1 .Names) ⊎ (is2 .Names)
(is1 ∪ is2) .rules = [ is1 .rules , is2 .rules ]

FCoInd⟦_,_⟧ : ∀{ℓc ℓp ℓn ℓn'} → (I : IS {ℓc} {ℓp} {ℓn} U)
  → (C : IS {ℓc} {ℓp} {ℓn'} U) → U → Set _
FCoInd⟦ I , C ⟧ = CoInd⟦ I ⊓ Ind⟦ I ∪ C ⟧ ⟧

SFCoInd⟦_,_⟧ : ∀{ℓc ℓp ℓn ℓn'} → (I : IS {ℓc} {ℓp} {ℓn} U)
  → (C : IS {ℓc} {ℓp} {ℓn'} U) → U → Size → Set _
SFCoInd⟦ I , C ⟧ = SCoInd⟦ I ⊓ Ind⟦ I ∪ C ⟧ ⟧
```

Figure 7.4: Interpretation generated by corules - datatype

sized types [Abel, 2012, Abel and Pientka, 2016, Abel et al., 2017] to ensure termination.

To allow compatibility with existing code implemented in either way, both versions in Remark 7.1.2 are supported by the library. In the first version, a possibly infinite proof tree for u is a record with only one field unfold containing an element of ISF [is] CoInd ⟦ is ⟧ u, that is, a proof that u can be derived by applying a rule from premises belonging to CoInd ⟦ is ⟧, which essentially consists of a rule with conclusion u and possibly infinite proof trees for its premises. In the second version, a possibly infinite proof tree is obtained by a **data** constructor, analogously to a finite one in the inductive interpretation; however, since proof trees are encoded as thunks, hence evaluated lazily, this encoding represents infinite trees as well. In other words, coinduction is "hidden" in the library type Thunk, which is a coinductive record with only one field force , intuitively representing the suspended computation.

The interpretation of a generalized inference system (see Figure 7.4) can then be encoded following exactly the definition in Section 1.2: it is the coinductive interpretation of I, restricted to rules whose conclusion is in the inductive interpretation of the (standard) inference system consisting of both rules I and corules C. The definition is provided in two flavours

```
ind[_] : ∀{ℓc ℓp ℓn ℓ}
    → (is : IS {ℓc} {ℓp} {ℓn} U)          -- IS
    → (S : U → Set ℓ)                      -- specification
    → ISClosed is S                        -- S is closed
    → Ind⟦ is ⟧ ⊆ S

coind[_] : ∀{ℓc ℓp ℓn ℓ}
    → (is : IS {ℓc} {ℓp} {ℓn} U)
    → (S : U → Set ℓ)
    → (S ⊆ ISF[ is ] S)          -- S is consistent
    → S ⊆ CoInd⟦ is ⟧

bounded-coind[_,_] : ∀{ℓc ℓp ℓn ℓn' ℓ}
    → (I : IS {ℓc} {ℓp} {ℓn} U)
    → (C : IS {ℓc} {ℓp} {ℓn'} U)
    → (S : U → Set ℓ)
    → S ⊆ Ind⟦ I ∪ C ⟧           -- S is bounded w.r.t. I ∪ C
    → S ⊆ ISF[ I ] S             -- S is consistent w.r.t. I
    → S ⊆ FCoInd⟦ I , C ⟧
```

Figure 7.5: Proof principles

where the coinductive interpretation is encoded by coinductive records and thunks, respectively, and uses two operators on inference systems, restriction ⊓ and union ∪. We report the codes in Figure 7.4. The former adds to each rule the side condition that the conclusion should satisfy P, as specified by the function addSideCond (here omitted). On the other hand, ∪ joins two inference systems.

The library also provides the proofs of relevant properties, e.g., that closed sets coincide with pre-fixed points, and consistent sets coincide with post-fixed points. Moreover, it is shown that the two versions of encoding of the coinductive interpretation (by coinductive records and thunks) are equivalent. Finally, the library provides the induction, coinduction, and bounded coinduction principles (see Propositions 1.2.1 to 1.2.3). We only report the statements in Figure 7.5 and we briefly recall their meaning.

- If S is closed, then each element of the inductively defined set Ind ⟦ is ⟧ satisfies S.

- If S is consistent, then each element satisfying S is in the coinductively defined set CoInd ⟦ is ⟧.

```
fcoind−to−ind : ∀{ℓc ℓp ℓn ℓn'}
    {is : IS {ℓc} {ℓp} {ℓn} U}{cois : IS {ℓc} {ℓp} {ℓn'} U}
    → FCoInd⟦ is , cois ⟧ ⊆ Ind⟦ is ∪ cois ⟧
```

Figure 7.6: Extract inductive proof

- If S is bounded, and consistent with respect to I, then each element which satisfies S belongs to the set FCoInd ⟦ I , C ⟧ defined by flexible coinduction.

Another useful theorem is that $\mathsf{Gen}[\![\mathcal{I},\mathcal{I}_{\mathsf{co}}]\!] \subseteq \mathsf{ind}\mathcal{I} \cup \mathcal{I}_{\mathsf{co}}$ (see Figure 7.6).

7.1.3 Basic Examples

We continue this section by showing how to use the library to define specific inference systems and prove their properties. In particular, we consider the basic examples in Section 1.2 that allow us to cover all the cases that we investigated before.

Example 7.1.1. *Consider the predicate member. We first recall its inference system and we give names to the (meta-)rules.*

$$\textsc{mem-h}\ \frac{}{member(x, x :: xs)} \qquad \textsc{mem-t}\ \frac{member(x, xs)}{member(x, y :: xs)}$$

The universe consists of pairs of elements and possibly infinite lists, implemented by the Agda library Colist *which uses thunks.*

```
U = A × Colist A ∞
data memberRN : Set where mem-h mem-t : memberRN

mem-h-r : FinMetaRule U
mem-h-r .Ctx = A × Thunk (Colist A) ∞
mem-h-r .comp (x , xs) =
        [] ,
        ──────────────────
        (x , x :: xs)

mem-t-r : FinMetaRule U
mem-t-r .Ctx = A × A × Thunk (Colist A) ∞
mem-t-r .comp (x , y , xs) =
          ((x , xs .force) :: []) ,
          ─────────────────────
          (x , y :: xs)

memberIS : IS U
memberIS .Names = memberRN
memberIS .rules mem-h = from mem-h-r
memberIS .rules mem-t = from mem-t-r
```

Here memberRN *are the rule names, and each rule name has an associated element of* FinMetaRule U*, which exactly encodes the meta-rule in the inference system at the beginning. Note, in* mem-t-r*, the use of the* force *field of* Thunk *to actually obtain the tail colist. This inference system is expected to define exactly the pairs* (x , xs) *such that* x *belongs to* xs*, that is, those satisfying the following specification*

```
memSpec : U → Set
memSpec (x , xs) = Σ[ i ∈ ℕ ] (Colist.lookup i xs ≡ just x)
```

where lookup : ℕ → Colist A ∞ → Maybe A *is the (standard) library function that returns the* i*-th element of* xs*, if any. As said in Section 1.2, to obtain the desired meaning this inference system has to be interpreted inductively, and soundness can be proved by the induction principle, that is, by providing a proof that the specification is closed with respect to the two meta-rules, as shown below.*

```
_member_ : A → Colist A ∞ → Set
x member xs = Ind⟦ memberIS ⟧ (x , xs)

memSpecClosed : ISClosed memberIS memSpec
memSpecClosed mem-h _ _ = zero , refl
memSpecClosed mem-t _ pr =
          let (i , proof) = pr Fin.zero in (suc i) , proof

memberSound : ∀ {x xs} → x member xs → memSpec (x , xs)
memberSound = ind[memberIS] memSpec memSpecClosed
```

For completeness there is no canonical technique; in this example, it can be proved by induction on the position (the index i *in the specification). For the complete proof see Ciccone [2020].* ⌟

Example 7.1.2. *Consider the predicate allPos from Section 1.2. We first* *recall its inference system and we give names to the (meta-)rules.*

$$\textsc{allP-}\Lambda\ \frac{}{\mathit{allPos}\,[]} \qquad \textsc{allP-t}\ \frac{\mathit{allPos}\,xs}{\mathit{allPos}\,x :: xs}\ x > 0$$

Then the universe consists of possibly infinite lists.

```
U : Set
U = Colist ℕ ∞
data allPosRN : Set where allP-Λ allP-t : allPosRN

allP-Λ-r : FinMetaRule U
allP-Λ-r .Ctx = ⊤
allP-Λ-r .comp c =
  [] ,
 ────────────────
  []

allP-t-r : FinMetaRule U
allP-t-r .Ctx = Σ[ (x , _) ∈ ℕ × Thunk (Colist ℕ) ∞ ] x > 0
allP-t-r .comp ((x , xs) , _) =
  ((xs .force) :: []) ,
 ────────────────
  (x :: xs)

allPosIS : IS U
allPosIS .Names = allPosRN
allPosIS .rules allP-Λ = from allP-Λ-r
allPosIS .rules allP-t = from allP-t-r
```

This inference system is expected to define exactly the lists such that all elements are positive, that is, those satisfying the following specification (where for simplicity, we use the predicate ∈, omitted, directly defined inductively). Notably, we could use member instead of ∈.

```
allPosSpec : U → Set
allPosSpec xs = ∀ {x} → x ∈ xs → x > 0
```

As said in Section 1.2, to obtain the desired meaning this inference system has to be interpreted coinductively, and completeness can be proved by the coinduction principle, that is, by providing a proof that the specification is consistent with respect to the inference system, as shown below.

```
allPos : U → Set
allPos = CoInd⟦ allPosIS ⟧

allPosSpecCons : ∀ {xs}
        → allPosSpec xs → ISF[ allPosIS ] allPosSpec xs
allPosSpecCons {[]} _ = allP-Λ , (tt , (refl , tt , λ ()))
allPosSpecCons {(x :: xs)} Sxs =
  allP-t ,
  ((x , xs) , (refl ,
              (Sxs here ,
              λ {Fin.zero → λ mem → Sxs (there mem)})))

allPosComplete : allPosSpec ⊆ allPos
allPosComplete = coind[ allPosIS ] allPosSpec allPosSpecCons
```

For what concerns the soundness, there is no canonical technique; in this example, when the colist is empty the proof that the specification holds is trivial. If the colist is not empty, then the proof proceeds by induction on the position of the element to be proved to be positive. For the complete proof see Ciccone [2020]. ⌟

Example 7.1.3. *Consider the predicate maxElem from Section 1.2. We recall once again its meta-(co)rules.*

$$\text{MAX-H}\quad \frac{}{\mathit{maxElem}(x, x :: [])} \qquad \text{MAX-T}\quad \frac{\mathit{maxElem}(x, xs)}{\mathit{maxElem}(\mathit{max}(x,y), y :: xs)} \qquad \text{CO-MAX-H}\quad \frac{}{\mathit{maxElem}(x, x :: xs)}$$

Then the universe consists of pairs of natural numbers and possibly infinite lists.

```
U : Set
U = ℕ × Colist ℕ ∞
data maxElemRN : Set where max-h max-t : maxElemRN
data maxElemCoRN : Set where co-max-h : maxElemCoRN

max-h-r : FinMetaRule U
max-h-r .Ctx =
  Σ[ (_ , xs) ∈ ℕ × Thunk (Colist ℕ) ∞ ] xs .force ≡ []
max-h-r .comp ((x , xs) , _) =
  [] ,
  ──────────────
  x , x :: xs

max-t-r : FinMetaRule U
max-t-r .Ctx =
  Σ[ (x , y , z , _) ∈
  ℕ × ℕ × ℕ × Thunk (Colist ℕ) ∞ ] z ≡ max x y
max-t-r .comp ((x , y , z , xs) , _) =
  (x , xs .force) :: [] ,
  ───────────────
  z , y :: xs

co-max-h-r : FinMetaRule U
co-max-h-r .Ctx = ℕ × Thunk (Colist ℕ) ∞
co-max-h-r .comp (x , xs) =
  [] ,
  ───────────────
  (x , x :: xs)

maxElemIS : IS U
maxElemIS .Names = maxElemRN
maxElemIS .rules max-h = from max-h-r
maxElemIS .rules max-t = from max-t-r

maxElemCoIS : IS U
maxElemCoIS .Names = maxElemCoRN
maxElemCoIS .rules co-max-h = from co-max-h-r
```

Note that in this example we have defined two inference systems, the rules and the corules. This generalized inference system is expected to define exactly the pairs (x , xs) *such that* x *is the maximal element of* xs*, that is, those satisfying the following specification, where to be the maximal element* x *should belong to* xs*, and be greater or equal than any* n *in* xs*.*

```
maxSpec inSpec geqSpec : U → Set
inSpec (x , xs) = x ∈ xs
geqSpec (x , xs) = ∀{n} → n ∈ xs → x ≡ max x n
maxSpec u = inSpec u × geqSpec u
```

As said in Section 1.2, the desired meaning is provided by the interpretation of the generalized inference system.

```
_maxElem_ : ℕ → Colist ℕ ∞ → Set
x maxElem xs = FCoInd⟦ maxElemIS , maxElemCoIS ⟧ (x , xs)
```

and the completeness can be proved by the bounded coinduction principle.

```
maxElemComplete : ∀{x xs} → maxSpec (x , xs) → x maxElem xs
maxElemComplete =
  bounded-coind[ maxElemIS , maxElemCoIS ] maxSpec
    (λ{(x , xs) → maxSpecBounded x xs})
    λ{(x , xs) → maxSpecCons x xs}
```

Notably, we have to prove that the specification is:

- bounded*, that is, contained in* _maxElem$_i$_ *, the inductive interpretation of the standard inference system consisting of both rules and corules, as shown below:*

```
_maxElemᵢ_ : ℕ → Colist ℕ ∞ → Set
x maxElemᵢ xs = Ind⟦ maxElemIS ∪ maxElemCoIS ⟧ (x , xs)

maxSpecBounded : ∀{x xs} → inSpec (x , xs)
  → geqSpec (x , xs) → x maxElemᵢ xs
```

- consistent *with respect to the inference system consisting of only rules, as shown below:*

```
maxSpecCons : ∀{x xs} → inSpec (x , xs) →
  geqSpec (x , xs) → ISF[ maxElemIS ] maxSpec (x , xs)
```

These proofs are omitted for the sake of brevity. See Ciccone [2020] for more details. Concerning the soundness there is no canonical technique. The proof can be split for the two components of the specification. It is worth noting that, for the soundness with respect to inSpec *, we first use* fcoind -to-ind *(see Figure 7.6), and then define* maxElemSound -in-ind*, omitted, by induction on the inference system consisting of rules and corules. The use of* fcoind -to-ind *in the proof corresponds to the fact that without corules unsound judgments could be derived.*

```
maxElemSound-in : ∀ {x xs} → x maxElem xs → inSpec (x , xs)
maxElem-sound-in max = maxElemSound-in-ind (fcoind-to-ind max)
```

Soundness with respect to geqSpec *is proved by induction on the position, that is, the proof of membership, of the element that must be proved to be less or equal. In this case, soundness would hold even in the purely coinductive case.* ⌟

Remark 7.1.3 (Code Duplication)*.* Of course, as Agda supports both inductive and coinductive dependent types, one could directly write Agda code for inductive, coinductive and even flexible coinductive definitions of concrete examples. We have explored this possibility in Ciccone [2020]. However, in this way, the definition is hard-wired with its semantics, and, for flexible coinduction, one has to manually construct the interpretation by combining in the correct way an inductive and a coinductive type and to prove the bounded coinduction principle for each example. For instance, the definition of maxElem will look as follows:

```
data _maxElem_ : ℕ → CoList ℕ ∞ → Size → Set where
  max-h : ∀ {x xs i} →force xs ≡ [] → x maxElem (x :: xs) i
  max-t : ∀ {x y xs i} → Thunk (x maxElem (force xs)) i
                        → z ≡ max x y
                        → z maxElem_i (y :: xs)
                        → z maxElem (y :: xs) i

data _maxElem_i_ : ℕ → CoList ℕ ∞ → Set where
  imax-h : ∀ {x xs} →force xs ≡ [] → x maxElem_i (x :: xs)
  imax-t : ∀ {x y xs} → x maxElem_i (force xs)) → z ≡ max x y
                      → z maxElem_i (y :: xs)
  co-max-h : ∀ {x xs} → x maxElem_i (x :: xs)
```

Clearly, this approach causes duplication of rules and code, as rules of the coinductive type have to be duplicated in the inductive one, making things rather complex. Our library instead hides all these details, exposing interfaces for interpretations and proof principles, so that the user only has to write code describing rules. ⌟

7.2 Divergence In The Lambda-Calculus

In Section 7.1.3 we formalized some basic examples to explain how the library can be used. In this section we describe a more significant example of instantiation: an inference system with corules providing a big-step semantics of lambda-calculus including divergence among the possible results Ancona et al. [2017a], reported in Figure 7.7. In this example, corules play a key role: indeed , considering, e.g., the divergent term $\Omega = (\lambda x.x\ x)\,(\lambda x.x\ x)$, in the standard inductive big-step semantics no result can be derived (an infinite proof tree is needed), as for a stuck term; in the purely coinductive interpretation, any judgment $\Omega \Downarrow v^\infty$ would be obtained Leroy and Grall [2009]. Since each node of the infinite proof tree for a judgment

$$\begin{array}{rcll} t & ::= & v \mid x \mid t_1\, t_2 \mid \ldots & \text{term} \\ v & ::= & \lambda x.t \mid \ldots & \text{value} \\ v^\infty & ::= & v \mid \infty & \text{result} \end{array} \qquad \textsc{coa}\ \frac{}{e \Downarrow \infty} \qquad \textsc{val}\ \frac{}{v \Downarrow v}$$

$$\textsc{app}\ \frac{t_1 \Downarrow \lambda x.t \quad t_2 \Downarrow v \quad t[x/v] \Downarrow v^\infty}{t_1\ t_2 \Downarrow v^\infty} \qquad \textsc{l-div}\ \frac{t_1 \Downarrow \infty}{t_1\ t_2 \Downarrow \infty} \qquad \textsc{r-div}\ \frac{t_1 \Downarrow v \quad t_2 \Downarrow \infty}{t_1\ t_2 \Downarrow \infty}$$

Figure 7.7: λ-calculus: syntax and big-step semantics

should also have a finite proof tree using the corules, the coaxiom [COA] forces to obtain only ∞ as result, see Ancona et al. [2017a] for a more detailed explanation. [2]

In rule [APP], v^∞ is used for the result, so the rule also covers the case when the evaluation of the body of the lambda abstraction diverges. As usual, $t[x/v]$ denotes capture-avoiding substitution. Rules [L-DIV] and [R-DIV] cover the cases when either t_1 or t_2 diverges, assuming a left-to-right evaluation strategy. Terms, values, and results are inductively defined, hence encoded by Agda datatypes. As customary in implementations of lambda-calculus, we use the De Bruijn notation: notably, `Term n` is the set of terms with `n` free variables.

```
data Term (n : ℕ) : Set where
  var : Fin n → Term n
  lambda : Term (suc n) → Term n
  app : Term n → Term n → Term n

data Value : Set where
  lambda : Term 1 → Value

term : Value → Term 0
term (lambda x) = lambda x

data Value∞ : Set where
  res : Value → Value∞
  ∞ : Value∞
```

The universe consists of big-step judgments (pairs consisting of a term and a result). The two inference systems of rules and corules are encoded

[2] Other examples of big-step semantic definitions with more sophisticated corules are given in Ancona et al. [2018, 2020].

below.

```
U : Set
U = Term 0 × Value∞

data BigStepRN : Set where val app l—div r—div : BigStepRN
data BigStepCoRN : Set where COA : BigStepCoRN

BigStepIS : IS U
BigStepIS .Names = BigStepRN
BigStepIS .rules val = from val-r
BigStepIS .rules app = from app-r
BigStepIS .rules L-DIV = from l-div-r
BigStepIS .rules R-DIV = from r-div-r

BigStepCoIS : IS U
BigStepCoIS .Names  = BigStepCoRN
BigStepCoIS .rules COA = from coa-r
```

where BigStepRN are the rule names, and each rule name has an associated element of FinMetaRule U. For instance, app-r is given below. The auxiliary function subst , omitted, implements capture-avoiding substitution.

```
app-r : FinMetaRule U
app-r .Ctx = Term 0 × Term 1 × Term 0 × Value × Value∞
app-r .comp (t1 , t , t2 , v , v∞) =
 (t1 , res (lambda t)) :: (t2 , res v) ::
 (subst t (term v) , v∞) :: [] ,
 ─────────────────────────
  (app t1 t2 , v∞)
```

The big-step semantics can be obtained as the interpretation of the generalized inference system, as shown below. We use the flavour with thunks.

```
_⇓_ : Term 0 → Value∞ → Size → Set
(t ⇓ v∞) i = SFCoInd⟦ BigStepIS , BigStepCoIS ⟧ (t , v∞) i

_⇓ᵢ_ : Term 0 → Value∞ → Set
t ⇓ᵢ v∞ = Ind⟦ BigStepIS ∪ BigStepCoIS ⟧ (t , v∞)
```

The second predicate (i stands for “inductive”) models that a judgment has a finite proof tree in the inference system consisting of rules and coaxiom, and will be used in proofs.

$$\beta\ \frac{}{(\lambda x.t)\ v \Rightarrow t[x/v]} \qquad \textsc{L-app}\ \frac{t_1 \Rightarrow t_1'}{t_1\ t_2 \Rightarrow t_1'\ t_2} \qquad \textsc{R-app}\ \frac{t_2 \Rightarrow t_2'}{v\ t_2 \Rightarrow v\ t_2'}$$

Figure 7.8: λ-calculus: small-step semantics

Small-step semantics, reported in Figure 7.8, can also be obtained appropriately instantiating the library. In this case, the universe consists of small-step judgments, which are pairs of terms. There is only one inference system, where `SmallStepRN` are the rule names, and each rule name has an associated element of `FinMetaRule U`.

```
U : Set
U = Term 0 × Term 0

data SmallStepRN : Set where β L-app R-app : SmallStepRN

SmallStepIS : IS U
SmallStepIS .Names = SmallStepRN
SmallStepIS .rules β = from β-r
SmallStepIS .rules L-app = from l-app-r
SmallStepIS .rules R-app = from r-app-r
```

For instance, β-r is given below.

```
β-r : FinMetaRule U
β-r .Ctx = Term 1 × Value
β-r .comp (t , v) =
  [] ,
  ──────────────────────────
  (app (lambda t) (term v) , subst t (term v))
```

The one-step relation $\Rightarrow$ is obtained as the inductive interpretation of the (standard) inference system. Then, finite computations are modeled by its reflexive and transitive closure $\Rightarrow^\star$, defined using `Star` in the Agda library, as shown below.

```
_⇒_ : Term 0 → Term 0 → Set
t ⇒ t' = Ind⟦ SmallStepIS ⟧ (t , t')

_⇒⋆_ : Term 0 → Term 0 → Set
_⇒⋆_ = Star _⇒_
```

Infinite computations, instead, are modeled by the relation $\Rightarrow^\infty$, coinductively defined by the meta-rule

$$\frac{t' \Rightarrow^\infty}{t \Rightarrow^\infty}\, t \Rightarrow t'$$

that we encoded in Agda by using thunks.

```
data _⇒∞ : Term 0 → Size → Set where
  step : ∀ {t t' i} → t ⇒ t' → Thunk (t' ⇒∞) i → t ⇒∞ i
```

The proof of equivalence between big-step and small-step semantics is structured as follows, where $\mathcal{S} = \{\langle t, v\rangle \mid t \Rightarrow^\star v\} \cup \{\langle t, \infty\rangle \mid t \Rightarrow^\infty\}$.

Soundness

$t \Downarrow v$ implies $t \Rightarrow^\star v$ We use fcoind -to-ind (see Figure 7.6), and then reason by induction on the judgment $t \Downarrow_{\mathsf{i}} v$. That is, we show that $t \Rightarrow^\star v$ is closed w.r.t. the inference system consisting of rules and corules. As already pointed out for the maxElem example, the use of fcoind -to-ind in the proof corresponds to the fact that, without the coaxiom [COA], unsound judgments would be derived, e.g., $\Omega \Downarrow v$ for $v \in \mathsf{Val}$.

$t \Downarrow \infty$ implies $t \Rightarrow^\infty$ This implication, instead, would hold even in the purely coinductive case. It can be proved from *progress* and *subject reduction* properties:

Progress $t \Downarrow \infty$ implies that there exists t' such that $t \Rightarrow t'$.

Subject reduction $t \Downarrow \infty$ and $t \Rightarrow t'$ implies $t' \Downarrow \infty$.

Completeness By bounded coinduction (see Proposition 1.2.3).

Boundedness

$t \Rightarrow^\star v$ implies $t \Downarrow_{\mathsf{i}} v$ By induction on the number of steps.

$t \Rightarrow^\infty$ implies $t \Downarrow_{\mathsf{i}} \infty$ Trivial, since the coaxiom coa can be applied.

Consistency We have to show that, for each $\langle t, v^\infty\rangle \in \mathcal{S}$, $\langle t, v^\infty\rangle$ is the consequence of a big-step rule where the premises are in $\mathcal{S}$ as well. We distinguish two cases.

$t \Rightarrow^\star v$ By induction on the number of steps. If it is 0, then t is a value, hence we can use rule [VAL]. Otherwise, t is an application, and we can use rule [APP].

$t \Rightarrow^\infty$ The term t is an application $t_1\, t_2$. We distinguish the following cases:

- t_1 diverges, hence we can use rule [L-DIV]
- t_1 converges and t_2 diverges, hence we can use rule [R-DIV]
- both t_1 and t_2 converge, hence we can use rule [APP].

Note that in this proof by cases we need to use the *excluded middle* principle, which is defined in the standard library, and postulated in our proof.

7.3 Indexed (Endo) Containers

We conclude this section by investigating the analogies and the differences of generalized inference systems with respect to *index containers*. *Indexed containers* [Altenkirch et al., 2015] are a rather general notion, meant to capture families of datatypes with some form of indexing. They are part of the Agda standard library. We report below the definition, simplified and adapted a little for presentation purpose. Notably, we use ad-hoc field names, chosen to reflect the explanation provided below.

```
record Container {ℓi ℓo}
 (I : Set ℓi) (O : Set ℓo) (ℓc ℓp : Level) : Set _ where
  constructor _ ◁ _/_
  field
    Cons  : (o : O) → Set ℓc
    Pos : ∀ {o} → Cons o → Set ℓp
    input : ∀ {o} (c : Cons o) → Pos c → I

⟦_⟧: ∀ {ℓi ℓo ℓc ℓp ℓ} {I : Set ℓi} {O : Set ℓo} →
      Container I O ℓc ℓp →
      (I → Set ℓ) → (O → Set _)
⟦ Cons ◁ Pos / input ⟧ X o =
      Σ[ c ∈ C o ] ((p : P c) → X (inp c p))
```

To explain the view of an inference system as an indexed container, we can think of the latter as describing a family of datatype constructors where I and O are input and output sorts, respectively. Then, Cons specifies, for each output sort o, the set of its constructors; for each constructor for o, Pos specifies a set of positions to store inputs to the constructor; finally, input specifies the input sort for each position in a constructor. The function ⟦_⟧ models the "semantics" of an indexed container, that is, given a family of inputs X indexed by I, it returns the family of outputs indexed by O which can be constructed by providing to some constructor inputs from P of correct sorts.

Then, inference systems can be defined as indexed containers where input and output sorts coincide, and are the elements of the universe, as follows.

```
ISCont : {ℓc ℓp : Level} → (U : Set ℓu) → Set _
ISCont {ℓc} {ℓp} U = Container U U ℓc ℓp
```

In this way, for each u : U:

- Cons u is the set of (indexes for) all the rules which have consequence u
- Pos c is the set of (indexes for) the premises of the c-th rule
- input c p is the p-th premise of the c-th rule

This view comes out quite naturally observing that an inference system is an element of $\wp(\wp(\mathcal{U})\times\mathcal{U})$; equivalently, a function which, for each $j\in\mathcal{U}$, returns the set of the sets of premises of all the rules with consequence j. In a constructive setting such as Agda, the powerset construction is not available, hence we have to use functions. So, for each element u, we need a type to index all rules with consequence u, and, for each rule, a type to index its premises, which are exactly the data of an indexed container. In other words, this view of inference systems as indexed containers explicitly interprets rules as constructors for proofs. Moreover, definitions in Section 1.2 can be easily obtained as instances of analogous definitions for indexed containers, building on the fact that the inference operator associated with an inference system turns out to be the semantics ⟦_⟧ of the corresponding container.

Whereas this encoding allows reuse of notions and code, a drawback is that information is structured in a rather different way from that "on paper"; notably, we group together rules with the same consequence, rather than those obtained as instances of the same "schema", that is, meta-rule. For this reason we developed the Agda library mimicking meta-rules. For instance, the inference system for *allPos* would be as follows:

```
allPosCont : ISCont (Colist ℕ ∞)
allPosCont .Cons [] = ⊤
allPosCont .Cons (x :: xs) = x > 0
allPosCont .Pos {[]} c = ⊥
allPosCont .Pos {x :: xs} c = Fin 1
allPosCont .input {x :: xs} c zero = xs .force
```

However, we can prove that the two notions are equivalent, as shown below. To this end, we define a translation C[_] from inference systems to indexed containers and a converse translation IS [_]. Note that in the translation C[_] each meta-rule is transformed in all its instantiations; more precisely, for each u : C, Cons u gives all the instantiations of meta-rules having u as consequence. Conversely, in the translation IS [_], each rule is transformed in a meta-rule with trivial context.

```
C[_] : ∀{ℓc ℓp ℓn} → IS {ℓc} {ℓp} {ℓn} U → Container U U _ ℓp
C[ is ] .Cons u = Σ[ rn ∈ is .Names ] Σ[ c ∈ is .rules rn .Ctx ]
    u ≡ is .rules rn .conclu c
C[ is ] .Pos (rn , _ , refl) = is .rules rn .Pos
C[ is ] .input (rn , c , refl) p = is .rules rn .prems c p

IS[_] : ∀{ℓc ℓp} →
          Container U U ℓc ℓp → IS {zero} {ℓp} {l ⊔ ℓc} U
IS[ C ] .Names = Σ[ u ∈ U ] C .Cons u
IS[ C ] .rules (u , c) =
  record {
    Ctx = ⊤ ;
    Pos = C .Pos c ;
    prems = λ _ r → C .input c r ;
    conclu = λ _ → u }

isf-to-c : ∀{ℓc ℓp ℓn ℓp} {is : IS {ℓc} {ℓp} {ℓn} U}
    {P : U → Set ℓp} → ISF[ is ] P ⊆ ⟦ C[ is ] ⟧ P
isf-to-c (rn , c , refl , pr) = (rn , c , refl) , pr

c-to-isf : ∀{l' ℓp ℓp} {C : Container U U l' ℓp}
    {P : U → Set ℓp} → ⟦ C ⟧ P ⊆ ISF[ IS[ C ] ] P
c-to-isf (c , pr) = (_ , c) , tt , refl , pr
```

Chapter 8

Properties Of Session Types

In this chapter we take into account three interesting properties of *binary* session types. All the properties under analysis mix safety and liveness. Hence, we use GISs (Section 1.2) as a reference framework for their definitions and proofs. Furthermore, for each property we first investigate its safety counterpart and we show how to obtain the desired one by using corules. Notably, all the results have been formalized in Agda [Ciccone and Padovani, 2021a,c, 2022a]. We chose to restrict to the binary case to simplify the mechanizations as much as possible.

The properties that we study are the following:

- *Fair Termination* of a session type. A fairly terminating session can always eventually reach termination
- *Fair Compliance* of two session types. Compliant sessions can always eventually interact to reach termination (asymmetric variant of Definition 2.3.1)
- *Fair Subtyping* between session types. This property is a compliance-preserving subtyping relation (see Definition 3.2.1)

We begin the chapter by mechanizing in Section 8.1 the notions presented in Section 2.2 and that will be used later. Then, in Section 8.2 we introduce an alternative definition of (binary) session types and its formalization. Then the chapter is split according to the property under analysis (Sections 8.3 to 8.5). Furthermore, for each property we show its Agda definition and we explain how the proof principles (see Section 1.2) can be used

```
open import Relation.Nullary using (¬_)
open import Relation.Unary using (Satisfiable; _∪_)
open import Relation.Binary.Core using (Rel)
open import Relation.Binary.Construct.Closure.ReflexiveTransitive
                 using (Star)
open import Function.Base using (_∘_)
```

Figure 8.1: Imported modules

to prove the correctness. At last, in Section 8.6 we detail the soundness and completeness proof of *fair compliance*. The formalization of all the results is available on GitHub [Ciccone and Padovani, 2021c].

Remark 8.0.1. In this chapter we refer to *binary session types* but we rely on a syntax different from the more conventional one presented in Section 1.3. This choice is mainly motivated by the fact we aimed at simplifying the mechanization as much as possible. In Section 8.2 we describe all the advantages of using such representation.

8.1 Fair Termination

In this section we mechanize *fair termination* in its general form, that is, considering an arbitrary labeled transition system. In particular, we mechanize the definitions and the notions that we presented in Section 2.2. We will instanciate such property in Sections 8.3 and 8.4 in session based scenarios. Notably, the entire development is available on GitHub (see Padovani [2022]).

Remark 8.1.1 (Imports). The codes that we present takes advantage of many features from the standard library. Figure 8.1 illustrates the modules that are imported and used in the formalization in this section. Satisfiable P holds when a proof of P can be exhibited whereas Star is used to obtain the reflexive transitive closure of the reference relation. ¬ P is equivalent to P → ⊥. For more details see Padovani [2022]. ⌟

First, we have to consider an arbitrary labelled transition system. Hence the Agda module is parametric on

```
( State : Set ) ( _~>_ : State → State → Set)
```

that is, a set of states and a transition relation. Then, we define the reflexive transitive closure of the transition relation and the set of predicates over states. The predicates Reduces and Stuck hold when a state can reduce to another one and when no reductions are allowed, respectively.

```
StateProp : Set₁
StateProp = State → Set

_~>*_ : Rel State _
_~>*_ = Star _~>_

Reduces : StateProp
Reduces S = Satisfiable (S ~>_)

Stuck : StateProp
Stuck S = ¬ (Reduces S)
```

A run is a maximal sequence of states S ~> S_1 ~> S_2 ~> ..., that is a sequence of reductions that is either infinite or it ends with a stuck state.

```
data Run (S : State) : Set
record ∞Run (S : State) : Set where
  coinductive
  field force : Run S

data Run S where
  stop : (stuck : Stuck S) → Run S
  _::_ : ∀{S'} (red : S ~> S') (ρ : ∞Run S') → Run S

RunProp : Set₁
RunProp = ∀{S} → Run S → Set
```

Note that runs are defined using a *coinductive record*. RunProp identifies predicates over runs. Now we model the fact that each property of states induces a corresponding property of runs in which there is a state that satisfies such property. A run is finite if it contains a stuck state.

```
data Eventually (P : StateProp) : RunProp where
  here : ∀{S} {ρ : Run S} (proof : P S) → Eventually P ρ
  next : ∀{S S'} (red : S ~> S') {ρ : ∞Run S'}
         (ev : Eventually P (ρ .force)) → Eventually P (red :: ρ)

Finite : RunProp
Finite = Eventually Stuck
```

A fairness assumption is a proposition over runs such that every partial run S $\sim>^*$ S' can be extended to a fair run. This condition is called feasibility or machine closure (see Lemma 2.2.1).

```
record FairnessAssumption : Set₁ where
  field
    Fair : RunProp
    feasible : ∀{S S'} (reds : S ~>* S') →
                       Σ[ ρ ∈ Run S' ] Fair (reds ++ ρ)

StuckFairness : FairnessAssumption
StuckFairness = record { Fair = Fair' ; feasible = feasible' }
  where
    Fair' : RunProp
    Fair' = Eventually (Stuck ∪ NonTerminating)
```

where ++ is used to concatenate runs and StuckFairness denotes our fairness assumption (see Definition 2.2.3). For the sake of clarity we omit the definition of feasible '. We recall that a run is fair if it contains finitely many weakly terminating states. This means that the run is either finite or divergent. Now we can define *fair termination* (see Definition 2.2.4). We recall that a state S is fairly terminating if the fair runs of S are finite.

```
WeaklyTerminating : StateProp
WeaklyTerminating S = Σ[ ρ ∈ Run S ] Finite ρ

FairlyTerminating : FairnessAssumption → StateProp
FairlyTerminating φ S = ∀{ρ : Run S} → Fair φ ρ → Finite ρ

Specification : StateProp
Specification S = ∀{S'} → S ~>* S' → WeaklyTerminating S'
```

Specification is the alternative characterization of fair termination that does not use fair runs. A state S satisfies the specification if any S' that is reachable from S is weakly terminating. A state is weakly terminating if it has a finite run. Now we can state Theorem 2.2.1 (proofs are omitted). In particular, we prove that the specification is a *necessary* condition for fair termination, regardless of the fairness assumption being made and that the specification is a *sufficient* condition for the notion of fair termination induced by our assumption.

```
ft→spec : (ϕ : FairnessAssumption) → ∀{S} →
        FairlyTerminating ϕ S → Specification S

spec→ft : ∀{S} →
        Specification S → FairlyTerminating StuckFairness S
```

At last, as a consequence we can prove that our assumption is the fairness assumption that induces the largest family of fairly terminating states.

```
ft→ft : (ϕ : FairnessAssumption) →
        ∀{S} → FairlyTerminating ϕ S →
        FairlyTerminating StuckFairness S
ft→ft ϕ = spec→ft ∘ ft→spec ϕ
```

8.2 Session Types

As noted in Remark 8.0.1, we rely on a different definition of (binary) session types with respect to that presented in Section 1.3. Although such choice has been originally motivated by the fact that we tried to simplify as much as possible the Agda mechanization of the properties (Sections 8.3 to 8.5), it also has the capability of encoding *dependent* session types, that is, types that depend on previously exchanged messages. We give more details later. We first introduce the new formulation of (binary) session types and then we detail the technical aspects of their formalization.

8.2.1 Session Types: An Alternative Formulation

We assume a set $\mathbb{V}$ of *message tags* that can be exchanged in communications. This set may include booleans, natural numbers, strings, and so forth. Hereafter, we assume that $\mathbb{V}$ contains at least *two* elements, otherwise branching protocols cannot be described and the theoretical development that follows becomes trivial. We use x, y, z to range over the elements of $\mathbb{V}$. We define the set $\mathbb{S}$ of *session types* over $\mathbb{V}$ using coinduction, to account for the possibility that session types (and the protocols they describe) may be infinite.

Definition 8.2.1 (Session Types)**.** Session types T, S are the possibly infinite trees coinductively generated by the productions

$$\begin{array}{rrcl} \textbf{Polarity} & \pi & \in & \{?, !\} \\ \textbf{Session type} & T, S & ::= & \mathsf{nil} \mid \pi\{x : T_x\}_{x \in \mathbb{V}} \end{array}$$

Hereafter, we write $\overline{\pi}$ for the opposite or dual polarity of π, that is $\overline{?} = !$ and $\overline{!} = ?$. Note that input and output session types specify continuations for *all* possible values in the set $\mathbb{V}$. The session type nil, which describes an *unusable* session channel, can be used as continuation for those values that *cannot* be received or sent. As we will see shortly, the presence of nil breaks the symmetry between inputs and outputs.

It is convenient to introduce some notation for presenting session types in a more readable and familiar form. Given a polarity π, a set $X \subseteq \mathbb{V}$ of values and a family $T_{x \in X}$ of session types, we let

$$\pi\{x : T_x\}_{x \in X} \overset{\mathsf{def}}{=} \pi\left(\{x : T_x\}_{x \in X} \cup \{x : \mathsf{nil}\}_{x \in \mathbb{V} \setminus X}\right)$$

so that we can omit explicit nil continuations. As a special case when all the continuations are nil, we write $\pi\mathsf{end}$ instead of $\pi\emptyset$. Both $?\mathsf{end}$ and $!\mathsf{end}$ describe session channels on which no further communications may occur, although they differ slightly with respect to the session types they can be safely combined with. Describing terminated protocols as degenerate cases of input/output session types reduces the amount of constructors needed for their Agda representation (see Section 8.2.2). Another common case for which we introduce a convenient notation is when the continuations are the same, regardless of the value being exchanged: in these cases, we write $\pi X.T$ instead of $\pi\{x : T\}_{x \in X}$. For example, $!\mathbb{B}.T$ describes a channel used for sending a boolean and then according to T and $?\mathbb{N}.S$ describes a channel used for receiving a natural number and then according to S. We abbreviate $\{x\}$ with x when no confusion may arise. So we write $!\mathsf{true}.T$ instead of $!\{\mathsf{true}\}.T$.

Finally, we define a partial operation $+$ on session types such that

$$\pi\{x : T_x\}_{x \in X} + \pi\{x : T_x\}_{x \in Y} \overset{\mathsf{def}}{=} \pi\{x : T_x\}_{x \in X \cup Y}$$

when $X \cap Y = \emptyset$. For example, $!\mathsf{true}.S_1 + !\mathsf{false}.S_2$ describes a channel used first for sending a boolean value and then according to S_1 or S_2 depending on the boolean value. It it easy to see that $+$ is commutative and associative and that $\pi\mathsf{end}$ is the unit of $+$ when used for combining session types with polarity π. Note that $T + S$ is undefined if the topmost polarities of T and S differ. We assume that $+$ binds less tightly than the '.' in continuations.

$$\textsc{input}\quad \frac{}{?x.S + T \xrightarrow{?x} S} \qquad\qquad \textsc{output}\quad \frac{}{!x.S + T \xrightarrow{!x} S}\ S \neq \mathsf{nil}$$

Figure 8.2: Labeled transition system

We do not introduce any concrete syntax for specifying infinite session types. Rather, we specify possibly infinite session types as solutions of equations of the form $S = \cdots$ where the metavariable S may also occur (guarded) on the right-hand side of '='. Guardedness guarantees that the session type S satisfying such equation does exist and is unique [Courcelle, 1983].

Example 8.2.1. *The session types T_1 and S_1 that satisfy the equations*

$$T_1 = !\mathsf{true}.!\mathbb{N}.T_1 + !\mathsf{false}.?\mathsf{end} \qquad S_1 = !\mathsf{true}.!\mathbb{N}^+.S_1 + !\mathsf{false}.?\mathsf{end}$$

both describe a channel used for sending a boolean. If the boolean is false*, the communication stops immediately (*?end*). If it is* true*, the channel is used for sending a natural number (a strictly positive one in S_1) and then according to T_1 or S_1 again. Notice how the structure of the protocol after the output of the boolean depends on the* value *of the boolean.*

The session types T_2 and S_2 that satisfy the equations

$$T_2 = ?\mathsf{true}.!\mathbb{N}.T_2 + ?\mathsf{false}.?\mathsf{end} \qquad S_2 = ?\mathsf{true}.!\mathbb{N}^+.S_2 + ?\mathsf{false}.?\mathsf{end}$$

differ from T_1 and S_1 in that the channel they describe is used initially for receiving *a boolean.* ⌟

We define the operational semantics of session types by means of a *labeled transition system*. *Labels*, ranged over by α, β, γ, have either the form $?x$ (input of message x) or the form $!x$ (output of message x). Transitions $T \xrightarrow{\alpha} S$ are defined in Figure 8.2

There is a fundamental asymmetry between send and receive operations: the act of sending a message is *active* – the sender may choose the message to send – while the act of receiving a message is *passive* – the receiver cannot cherry-pick the message being received. We model this asymmetry with the side condition $S \neq \mathsf{nil}$ in [OUTPUT] and the lack thereof in [INPUT]: a process that uses a session channel according to $!\{x : T_x\}_{x\in\mathbb{V}}$ refrains from sending a message x if $T_x = \mathsf{nil}$, namely if the channel becomes unusable by sending that particular message, whereas a process that uses a session channel according to $?\{x : T_x\}_{x\in\mathbb{V}}$ cannot decide which message x it will

$$\textsc{sync}\quad \frac{}{S \# T \to S' \# T'}\; S \xrightarrow{\overline{\alpha}} S', T \xrightarrow{\alpha} T'$$

Figure 8.3: Reduction of session

receive, but the session channel becomes unusable if an unexpected message arrives. The technical reason for modeling this asymmetry is that it allows us to capture a realistic communication semantics. For the time being, these transition rules allow us to appreciate a little more the difference between !end and ?end. While both describe a session endpoint on which no further communications may occur, !end is "more robust" than ?end since it has no transitions, whereas ?end is "more fragile" than !end since it performs transitions, all of which lead to nil. For this reason, we use !end to flag successful session termination, whereas ?end only means that the protocol has ended.

To describe *sequences* of consecutive transitions performed by a session type we use another relation $\overset{\varphi}{\Longrightarrow}$ where φ and ψ range over strings of labels. As usual, ε denotes the empty string and juxtaposition denotes string concatenation. The relation $\overset{\varphi}{\Longrightarrow}$ is the least one such that $T \overset{\varepsilon}{\Longrightarrow} T$ and if $T \xrightarrow{\alpha} S$ and $S \overset{\varphi}{\Longrightarrow} R$, then $T \overset{\alpha\varphi}{\Longrightarrow} R$.

At last, we need to model the evolution of a session as client and server interact. To this aim, we represent a session as a pair $R \# T$ where R describes the behavior of the client and T that of the server. Sessions reduce according to the rule in Figure 8.3 where $\overline{\alpha}$ is the *complementary action* of α defined by $\overline{\pi x} = \overline{\pi} x$. We extend $\overline{\cdot}$ to traces in the obvious way and we write $\Rightarrow$ for the reflexive, transitive closure of $\to$. We write $R \# T \to$ if $R \# T \to R' \# T'$ for some R' and T' and $R \# T \not\to$ if not $R \# T \to$.

8.2.2 Agda Mechanization of Types

𝄢 We postulate the existence of $\mathbb{V}$, representing the set of values that can be exchanged in communications. The actual Agda formalization is parametric on an arbitrary set $\mathbb{V}$, the only requirement being that $\mathbb{V}$ must be equipped with a decidable notion of equality. We will make some assumptions on the nature of $\mathbb{V}$ when we present specific examples. To begin the formalization, we declare the two data types we use to represent session types. Because these types are mutually recursive, we declare them in advance so that we can later refer to them from within the definition of each.

```
data SessionType : Set
record ∞SessionType : Set

data SessionType where
  nil : SessionType
  inp out : Continuation → SessionType

record ∞SessionType where
  coinductive
  field force : SessionType
```

The SessionType data type provides three constructors corresponding to the three forms of a session type (see Definition 8.2.1). The ∞ SessionType wraps a session type within a coinductive record, so as to make it possible to represent *infinite* session types. The record has just one field force that can be used to access the wrapped session type. By opening the record, we make the force field publicly accessible without qualifiers.

Remark 8.2.1. Coinductive records are the built-in technique for dealing with infinite datatypes. It has been already used in Chapter 7 and Section 8.3 for implementing the coinductive interpretations of an inference system and possibly infinite runs in a labeled transitions system. Although the *sized types* approach leads to syntactically easier datatypes, a recent Agda update highlighted an inconsistency when using both approaches together. Since *thunks*, on which sized types are based on, are defined as a coinductive record, we decided to mainly rely on the former approach for the sake of simplicity. ⌟

A key design choice of our formalization is the representation of continuations $\{x : S_x\}_{x \in \mathbb{V}}$, which may have infinitely many branches if $\mathbb{V}$ is infinite. We represent continuations in Agda as *total functions* from $\mathbb{V}$ to session types, thus:

```
Continuation : Set
Continuation = 𝕍 → ∞SessionType
```

Example 8.2.2. *Consider once again the session type T_1 discussed in Example 8.2.1:*

$$T_1 = !\mathsf{true}.!\mathbb{N}.T_1 + !\mathsf{false}.?\mathsf{end}$$

In this case we assume that $\mathbb{V}$ is the disjoint sum of $\mathbb{N}$ (the set of natural numbers) and $\mathbb{B}$ (the set of boolean values) with constructors nat *and* bool *. It is useful to also define once and for all the* continuation empty *, which maps every message to* nil*:*

```
empty : Continuation
empty _ .force = nil
```

Now, the Agda encoding of T_1 is shown below:

```
T₁ : SessionType
T₁ = out f
  where
    f g : Continuation
    f (nat _) .force = nil
    f (bool true) .force = out g
    f (bool false) .force = inp empty
    g (nat _) .force = out f
    g (bool _) .force = nil
```

The continuations f *and* g *are defined using pattern matching on the message argument and using copattern matching to specify the value of the* force *field of the resulting coinductive record. They represent the two stages of the protocol:* f *allows sending a boolean (but no natural number) and, depending on the boolean, it continues as* g *or it terminates;* g *allows sending a natural number (but no boolean) and continues as* f*.* ⌟

Example 8.2.2 illustrates a simple form of *dependency* whereby the structure of a communication protocol may depend on the content of previously exchanged messages. The fact that we use Agda to write continuations means that we can model sophisticated forms of dependencies that are found only in the most advanced theories of dependent session types [Toninho et al., 2011, Toninho and Yoshida, 2018, Thiemann and Vasconcelos, 2020, Ciccone and Padovani, 2020].

Example 8.2.3. *Consider the Agda encoding of a session type*

$$!(n : \mathbb{N}).\underbrace{!\mathbb{B}\ldots!\mathbb{B}}_{n}.!\mathsf{end}$$

describing a channel used for sending a natural number n followed by n boolean values:

```
BoolVector : SessionType
BoolVector = out g
  where
    f : ℕ → Continuation
    f zero _ .force = nil
    f (suc n) (bool _) .force = out (f n)
    f (suc n) (nat _) .force = nil
    g : Continuation
    g (nat n) .force = out (f n)
    g (bool _) .force = nil
```

We will not discuss further examples of dependent session types. However, note that the possibility of encoding protocols such as BoolVector *has important implications on the scope of our study: it means that the results we have presented and formally proved in Agda hold for a large family of session types that includes dependent ones.* ⌟

We now provide a few auxiliary predicates on session types and continuations. First of all, we say that a session type is *defined* if it is different from nil:

```
data Defined : SessionType → Set where
  inp : ∀{f} → Defined (inp f)
  out : ∀{f} → Defined (out f)
```

Concerning continuations, we define the *domain* of a continuation function f to be the subset of $\mathbb{V}$ such that $f\ x$ is defined, that is dom $f = \{x \in \mathbb{V} \mid f\ x \neq \text{nil}\}$. We say that a continuation is *empty* if so is its domain. On the contrary, a non-empty continuation is said to have a *witness*. We define a Witness predicate to characterize this condition.

```
dom : Continuation → Pred 𝕍 Level.zero
dom f x = Defined (f x .force)

EmptyContinuation : Continuation → Set
EmptyContinuation f = Relation.Unary.Empty (dom f)

Witness : Continuation → Set
Witness f = Relation.Unary.Satisfiable (dom f)
```

where Satisfiable P and Empty P mean that at least an element satisfies P and no elements satisfy P, respectively.

Example 8.2.4. *Consider the* empty *continuation defined in Example 8.2.2. We can prove that it is indeed an empty one.*

```
empty-is-empty : EmptyContinuation empty
empty-is-empty _ ()
```

⌟

We now define a predicate Win to characterize the session type !end and End to characterize πend:

```
data Win : SessionType → Set where
  out : ∀{f} → EmptyContinuation f → Win (out f)

data End : SessionType → Set where
  inp : ∀{f} (U : EmptyContinuation f) → End (inp f)
  out : ∀{f} (U : EmptyContinuation f) → End (out f)
```

At last, we need a representation of sessions as pairs $R \# S$ of session types. To this aim, we introduce the Session data type as an alias for pairs of session types.

```
Session : Set
Session = SessionType × SessionType
```

8.2.3 Agda Mechanization of LTS

Let us move onto the definition of transitions for session types. We begin by defining an Action data type to represent input/output actions, which consist of a polarity and a value. The *complementary action* of α, denoted by $\overline{\alpha}$ in the previous sections, is computed by the function co- action .

```
data Action : Set where
  I O : 𝕍 → Action

co-action : Action → Action
co-action (I x) = O x
co-action (O x) = I x
```

A *transition* is a ternary relation among two session types and an action. A value of type `Transition S α T` represents the transition $S \xrightarrow{\alpha} T$. Note that the premise `x ∈ dom f` in the constructor `out` corresponds to the side condition $S \neq$ nil of rule [OUTPUT].

```
data Transition : SessionType → Action → SessionType → Set
  where
    inp : ∀{f x} → Transition (inp f) (I x) (f x .force)
    out : ∀{f x} →
        x ∈ dom f → Transition (out f) (O x) (f x .force)
```

A session *reduces* when client and server synchronize, by performing actions with opposite polarities and referring to the same message. We formalize the reduction relation in Figure 8.2 as the `Reduction` data type, so that a value of type `Reduction` $(R \# S)$ $(R' \# S')$ witnesses the reduction $R \# S \rightarrow R' \# S'$. At last, the weak reduction relation is called `Reductions` and is defined as the reflexive, transitive closure of `Reduction`, just like $\Rightarrow$ is the reflexive, transitive closure of $\rightarrow$. We make use of the `Star` data type from Agda's standard library to define such closure.

```
data Reduction : Session → Session → Set where
  sync : ∀{α R R' S S'} → Transition R (co−action α) R' →
         Transition S α S' → Reduction (R , S) (R' , S')

Reductions : Session → Session → Set
Reductions = Star Reduction
```

8.3 Fair Termination

In this section we characterize fair termination of a session type according to its general formulation in Theorem 2.2.1. We say that a session type is fairly terminating if it preserves the possibility of reaching !end or ?end along all of its transitions that do not lead to nil. Fair termination of S does not necessarily imply that there exists an upper bound to the length of communications that follow the protocol S, but it guarantees the absence of "infinite loops" whereby the communication is forced to continue forever.

8.3.1 Definition

To formalize fair termination we need the notion of *trace*, which is a finite sequence of actions performed on a session channel while preserving usability of the channel.

Definition 8.3.1 ((Maximal) traces)**.** The *traces* of a session S are defined as $\mathtt{tr}(S) \overset{\mathsf{def}}{=} \{\varphi \mid \exists T : S \overset{\varphi}{\Longrightarrow} T \neq \mathsf{nil}\}$. We say that $\varphi \in \mathtt{tr}(S)$ is *maximal* if $\varphi\psi \in \mathtt{tr}(S)$ implies $\psi = \varepsilon$.

Remark 8.3.1. Definition 8.3.1 differs from the $\mathsf{paths}()$ of a session type (Definition 3.2.7) since we require that the involved session type does not reduce to nil. This difference is motivated by the different definition of session types. $\lrcorner$

Example 8.3.1. *We have* $\mathtt{tr}(\mathsf{nil}) = \emptyset$ *and* $\mathtt{tr}(!\mathsf{end}) = \mathtt{tr}(?\mathsf{end}) = \{\varepsilon\}$*. Note that* $!\mathsf{end}$ *and* $?\mathsf{end}$ *have the same traces but different transitions (hence different behaviors).* $\lrcorner$

A *maximal trace* is a trace that cannot be extended any further.

Definition 8.3.2 (Fair Termination)**.** We say that S is *fairly terminating* if, for every $\varphi \in \mathtt{tr}(S)$, there exists ψ such that $\varphi\psi \in \mathtt{tr}(S)$ and $\varphi\psi$ is maximal.

Example 8.3.2. ε *is a maximal trace of both* $!\mathsf{end}$ *and* $?\mathsf{end}$ *but not of* $!\mathbb{B}.?\mathsf{end}$ *whereas* $!\mathsf{true}$ *and* $!\mathsf{false}$ *are maximal traces of* $!\mathbb{B}.?\mathsf{end}$*.* $\lrcorner$

Example 8.3.3. *All of the session types presented in Example 8.2.1 are fairly terminating. The session type* $R = !\mathbb{B}.R$*, which describes a channel used for sending an infinite stream of boolean values, is not fairly terminating because no trace of* R *can be extended to a maximal one. Note that also* $R' \overset{\mathsf{def}}{=} !\mathsf{true}.R + !\mathsf{false}.!\mathsf{end}$ *is not fairly terminating, even though there is a path leading to* $!\mathsf{end}$*, because fair termination must be* preserved *along all possible transitions of the session type, whereas* $R' \xrightarrow{!\mathsf{true}} R$ *and* R *is not fairly terminating. Finally,* nil *is trivially fairly terminating because it has no trace.* $\lrcorner$

To find an inference system for fair termination observe that the set $\mathbb{F}$ of fairly terminating session types is the largest one that satifies the following two properties:

1. it must be possible to reach either $!\mathsf{end}$ or $?\mathsf{end}$ from every $S \in \mathbb{F} \setminus \{\mathsf{nil}\}$;

$$\text{T-NIL}\ \frac{}{\mathsf{nil}{\Downarrow}} \qquad \text{T-ALL}\ \frac{\forall x \in \mathbb{V} : T_x{\Downarrow}}{\pi\{x : T_x\}_{x\in\mathbb{V}}{\Downarrow}} \qquad \text{T-ANY}\ \frac{S{\Downarrow}}{\pi x.S + T}\ S \neq \mathsf{nil}$$

Figure 8.4: Generalized inference system $\langle \mathcal{T}, \mathcal{T}_{\mathsf{co}} \rangle$ for fair termination

2. the set $\mathbb{F}$ must be closed by transitions, namely if $S \in \mathbb{F}$ and $S \xrightarrow{\alpha} T$ then $T \in \mathbb{F}$.

Neither of these two properties, taken in isolation, suffices to define $\mathbb{F}$: the session type R' in Example 8.3.3 enjoys property (1) but is not fairly terminating; the set $\mathbb{S}$ is obviously the largest one with property (2), but not every session type in it is fairly terminating. This suggests the definition of $\mathbb{F}$ as the largest subset of $\mathbb{S}$ satisfying (2) and whose elements are *bounded* by property (1), which is precisely what corules allow us to specify.

Figure 8.4 shows a GIS $\langle \mathcal{T}, \mathcal{T}_{\mathsf{co}} \rangle$ for fair termination with the usual notation for single-lined rules and doubly-lined corules. The axiom [T-NIL] indicates that nil is fairly terminating in a trivial way (it has no trace), while [T-ALL] indicates that fair termination is closed by all transitions. Note that these two rules, interpreted coinductively, are satisfied by all session types, hence $\{S \mid S{\Downarrow} \in \mathsf{CoInd}[\![\mathcal{S}]\!]\} = \mathbb{S}$.

Theorem 8.3.1. *T is fairly terminating if and only if $S{\Downarrow} \in \mathsf{Gen}[\![\mathcal{S}, \mathcal{S}_{\mathsf{co}}]\!]$.*

Proof sketch. For the "if" part, suppose $S{\Downarrow} \in \mathsf{Gen}[\![\mathcal{S}, \mathcal{S}_{\mathsf{co}}]\!]$ and consider a trace $\varphi \in \mathtt{tr}(S)$. That is, $S \overset{\varphi}{\Longrightarrow} T$ for some $T \neq \mathsf{nil}$. Using [T-ALL] we deduce $T{\Downarrow} \in \mathsf{Gen}[\![\mathcal{S}, \mathcal{S}_{\mathsf{co}}]\!]$ by means of a simple induction on φ. Now $T{\Downarrow} \in \mathsf{Gen}[\![\mathcal{S}, \mathcal{S}_{\mathsf{co}}]\!]$ implies $T{\Downarrow} \in \mathsf{Ind}[\![\mathcal{S} \cup \mathcal{S}_{\mathsf{co}}]\!]$. Another induction on the (well-founded) derivation of this judgment, along with the witness message x of [T-ANY], allows us to find ψ such that $\varphi\psi$ is a maximal trace of S.

For the "only if" part, we apply the bounded coinduction principle (see Proposition 1.2.3). Since we have already argued that the coinductive interpretation of the GIS in Figure 8.4 includes all session types, it suffices to show that S fairly terminating implies $S{\Downarrow} \in \mathsf{Ind}[\![\mathcal{S} \cup \mathcal{S}_{\mathsf{co}}]\!]$. From the assumption that S is fairly terminating we deduce that there exists a maximal trace $\varphi \in \mathtt{tr}(S)$. An induction on φ allows us to derive $S{\Downarrow}$ using repeated applications of [T-ANY], one for each action in φ, topped by a single application of [T-NIL]. $\square$

8.3.2 Agda Formalization

We now use the Agda library for GIS (see Chapter 7) to formally define the inference system for fair termination shown in Figure 8.4. First, we encode the universe on which the predicate is defined. In this case, it consists of session types, that is, $\mathbb{S}$.

```
U : Set
U = SessionType
```

Now we define the sets containing the names of the (co)rules. Although we try to be as consistent as possible with respect to the GIS in Figure 8.4, we need to split both [T-ALL] and [T-ANY] according to the polarity of the involved session type. Hence, we obtain three rules and two corules.

```
data RuleNames : Set where
  nil inp out : RuleNames

data CoRuleNames : Set where
  inp out : CoRuleNames
```

We can look at the definitions of the five metarules. Notably, as we presented in Chapter 7, the library offers a simpler way for defining a metarule with a finite number of premises (see FinMetaRule datatype). We take advantage of such feature to define the axiom as well as the corules that differ from the rules since they have a single premise. We show rules and corules separately.

```
nil-r : FinMetaRule U
nil-r .Ctx = ⊤
nil-r .comp _ =
  [] ,
 ─────
  nil

inp-r : MetaRule U
inp-r .Ctx = Continuation
inp-r .Pos _ = 𝕍
inp-r .prems f p = f p .force
inp-r .conclu f = inp f

out-r : MetaRule U
out-r .Ctx = Continuation
```

```
out-r .Pos _ = 𝕍
out-r .prems f p = f p .force
out-r .conclu f = out f
```

The axiom simply states that nil is trivially fairly terminating. Concerning the context, we use the datatype ⊤ from the standard library which can be always instantiated with the constructor tt . The two rules have have the Pos field set to 𝕍 since they have a premise for each element in 𝕍. Note also that each session type is unfolded in the premises by accessing the force filed of the coinductive record ∞ SessionType .

```
inp-co-r : FinMetaRule U
inp-co-r .Ctx = Σ[ (f , x) ∈ Continuation × 𝕍 ] x ∈ dom f
inp-co-r .comp ((f , x) , _) =
  f x .force :: [] ,
  ─────────────────────
  inp f

out-co-r : FinMetaRule U
out-co-r .Ctx = Σ[ (f , x) ∈ Continuation × 𝕍 ] x ∈ dom f
out-co-r .comp ((f , x) , _) =
  f x .force :: [] ,
  ─────────────────────
  out f
```

As mentioned before, the corules differ from the rules because they have a single premise. This is represented by the existential quantifier in the context which informally asks that the must be a continuation of the involved session type which is fairly terminating. Now we can compose the two inference systems FairTerminationIS and FairTerminationCOIS consisting of the rules and the corules, respectively. The desired predicate FairTermination is obtained through the generalized interpretation of the whole inference system. FairTerminationI is the inductive predicate obtained by inductively interpreting all the rules.

```
FairTerminationIS : IS U
Names FairTerminationIS = RuleNames
rules FairTerminationIS nil = from nil-r
rules FairTerminationIS inp = inp-r
rules FairTerminationIS out = out-r

FairTerminationCOIS : IS U
FairTerminationCOIS .Names = CoRuleNames
FairTerminationCOIS .rules inp = from inp−co−r
FairTerminationCOIS .rules out = from out−co−r

FairTermination : SessionType → Set
FairTermination =
  FCoInd⟦ FairTerminationIS , FairTerminationCOIS ⟧

FairTerminationI : SessionType → Set
FairTerminationI =
  Ind⟦ FairTerminationIS ∪ FairTerminationCOIS ⟧
```

where we recall that the function `from` turns a `FinMetaRule` into a `MetaRule` (see Figure 7.2).

In order to prove the correctness of `FairTermination` we need to encode the specification, that is, Definition 8.3.2.

```
FairTerminationS : SessionType → Set
FairTerminationS S = ∀{ϕ} →
  ϕ ∈ ⟦ S ⟧ → ∃[ ψ ] (ϕ ++ ψ ∈ Maximal ⟦ S ⟧)
```

where ϕ,ψ : `Trace` and traces are defined as `List Action`. Hence `_++_` is the library function for computing the concatenation of two lists. ⟦_⟧ denotes the set of all the possible traces of a session type. Finally, `Maximal` is a predicate on sets of traces and denotes all those traces that cannot be extended (see Definition 8.3.1).

The correctness of `FairTermination` is expressed in terms of soundness and completeness with respect to `FairTerminationS`. We are not going to detail the proofs. Instead we report the declarations of the main lemmas. Concerning the *soundness*, GISs do not provide a canonical technique to prove it.

```
sound : FairTermination ⊆ FairTerminationS
```

where `P ⊆ Q` is equivalent to `∀{x} → P x → Q x` with `P`, `Q` predicates over some `A : Set _` and `x : A`. For what concerns the *completeness*, it is by bounded coinduction (Proposition 1.2.3, Figure 7.5); hence we have to prove that `FairTerminationS` is *bounded* and *consistent* with respect to the GIS.

```
bounded : FairTerminationS ⊆ FairTerminationI

consistent :
  FairTerminationS ⊆ ISF[ FairTerminationIS ] FairTerminationS

complete : FairTerminationS ⊆ FairTermination
complete =
  bounded−coind[ FairTerminationIS , FairTerminationCOIS ]
    FairTerminationS bounded consistent
```

Remark 8.3.2. Assume that we instanciate Specification from Section 8.1 using SessionType (see Section 8.2.2) as set of states and Transition (see Section 8.2.3) as transition system. Such instance of Specification will not be equivalent to FairTerminationS as, for example, a client sending always true would be fairly terminating since the false branch would always lead to nil. To make the two specifications equivalent we need to instanciate Specification by using the following transition system

$$\frac{S \xrightarrow{\alpha} T}{S \xrightarrow{\alpha}{}' T}\ T \neq \mathsf{nil}$$

⌟

8.4 Fair Compliance

In this section we define and characterize two *compliance* relations for session types, which formalize the "successful" interaction between a client and a server connected by a session. The notion of "successful interaction" that we consider is biased towards client satisfaction, but see Remark 8.4.1 for a discussion about alternative notions.

8.4.1 Definition

The first compliance relation that we consider requires that, if the interaction in a session stops, it is because the client "is satisfied" and the server "has not failed" (recall that a session type can turn into nil only if an unexpected message is received). Formally:

Definition 8.4.1 (Compliance)**.** We say that R is *compliant* with T if $R \# T \Rightarrow R' \# T' \nrightarrow$ implies $R' = !\mathsf{end}$ and $T' \neq \mathsf{nil}$.

This notion of compliance is an instance of *safety property* in which the invariant being preserved at any stage of the interaction is that either client and server are able to synchronize further, or the client is satisfied and the server has not failed.

The second compliance relation that we consider adds a *liveness* requirement namely that, no matter how long client and server have been interacting with each other, it is always possible to reach a configuration in which the client is satisfied and the server has not failed.

Definition 8.4.2 (Fair Compliance)**.** We say that R is *fairly compliant* with T if $R \# T \Rightarrow R' \# T'$ implies $R' \# T' \Rightarrow \mathsf{!end} \# T''$ with $T'' \neq \mathsf{nil}$.

Notably, fair compliance corresponds to a *successful* form of fair termination. It is easy to show that fair compliance implies compliance, but there exist compliant session types that are not fairly compliant, as illustrated in the following example.

Example 8.4.1. *Recall Example 8.2.1 and consider the session types R_1 and R_2 such that*

$$R_1 = ?\mathsf{true}.?\mathbb{N}.R_1 + ?\mathsf{false}.\mathsf{!end} \qquad R_2 = \mathsf{!true}.(?0.\mathsf{!end} + ?\mathbb{N}^{+}.R_2)$$

Then R_1 is fairly compliant with both T_1 and S_1 and R_2 is compliant with both T_2 and S_2. Even if S_1 exhibits fewer behaviors compared to T_1 (it never sends 0 to the client), at the beginning of a new iteration it can always send false *and steer the interaction along a path that leads R_1 to success. On the other hand, R_2 is fairly compliant with T_2 but not with S_2. In this case, the client insists on sending* true *to the server in hope to receive 0, but while this is possible with the server T_2, the server S_2 only sends strictly positive numbers.*

This example also shows that fair termination of both client and server is not sufficient, in general, to guarantee fair compliance. Indeed, both R_2 and S_2 are fairly terminating, but they are not fairly compliant. The reason is that the sequences of actions leading to !end *on the client side are not necessarily the same (complemented) traces that lead to π*end *on the server side. Fair compliance takes into account the synchronizations that can actually occur between client and server.* ⌟

Remark 8.4.1. With the above notions of compliance we can now better motivate the asymmetric modeling of (passive) inputs and (active) outputs in the labeled transition system of session types (Figure 8.2). Consider the session types $R = \mathsf{!true}.\mathsf{!end} + \mathsf{!false}.\mathsf{!false}.\mathsf{!end}$ and $S = ?\mathsf{true}.?\mathsf{end}$. Note that R describes a client that succeeds by either sending a single true value or by sending two false values in sequence, whereas S describes a server

C-SUCCESS
$$\frac{}{\mathsf{!end} \dashv T}\ T \neq \mathsf{nil}$$

C-INP-OUT
$$\frac{\forall x \in X : S_x \dashv T_x}{?\{x : S_x\}_{x \in \mathbb{V}} \dashv !\{x : T_x\}_{x \in X}}\ X \neq \emptyset$$

C-SYNC
$$\frac{S \dashv T}{\pi x.S + S' \dashv \overline{\pi} x.T + T'}$$

C-OUT-INP
$$\frac{\forall x \in X : S_x \dashv T_x}{!\{x : S_x\}_{x \in X} \dashv ?\{x : T_x\}_{x \in \mathbb{V}}}\ X \neq \emptyset$$

Figure 8.5: Generalized inference system $\langle \mathcal{C}, \mathcal{C}_{\mathsf{co}} \rangle$ for fair compliance

that can only receive a single true value. If we add the same side condition $S \neq \mathsf{nil}$ also for [INPUT] then R would be compliant with S. Indeed, the server would be unable to perform the $?\mathsf{false}$-labeled transition, so that the only synchronization possible between R and S would be the one in which true is exchanged. In a sense, with the $S \neq \mathsf{nil}$ side condition in [INPUT] we would be modeling a communication semantics in which client and server *negotiate* the message to be exchanged depending on their respective capabilities. Without the side condition, the message to be exchanged is always chosen by the active part (the sender) and, if the passive part (the receiver) is unable to handle it, the receiver fails. The chosen asymmetric communication semantics is also key to induce a notion of (fair) subtyping that is *covariant* with respect to inputs (see Section 8.4). ⌟

Figure 8.5 presents the GIS $\langle \mathcal{C}, \mathcal{C}_{\mathsf{co}} \rangle$ for fair compliance. Intuitively, a derivable judgment $S \dashv T$ means that the client S is (fairly) compliant with the server T. Rule [C-SUCCESS] relates a satisfied client with a non-failed server. Rules [C-INP-OUT] and [C-OUT-INP] require that, no matter which message is exchanged between client and server, the respective continuations are still fairly compliant. The side condition $X \neq \emptyset$ guarantees progress by making sure that the sender is capable of sending at least one message. As we will see, the coinductive interpretation of $\mathcal{C}$, which consists of these three rules, completely characterizes compliance (Definition 8.4.1). However, these rules do not guarantee that the interaction between client and server can always reach a successful configuration as required by Definition 8.4.2. For this, the corule [C-SYNC] is essential. Indeed, a judgment $S \dashv T$ that is derivable according to the generalized interpretation of the GIS $\langle \mathcal{C}, \mathcal{C}_{\mathsf{co}} \rangle$ must admit a well-founded derivation tree also in the inference system $\mathcal{C} \cup \mathcal{C}_{\mathsf{co}}$. Since [C-SUCCESS] is the only axiom in this inference system, finding a well-founded derivation tree in $\mathcal{C} \cup \mathcal{C}_{\mathsf{co}}$ boils down to finding a (finite) path of synchronizations from $S \# T$ to a successful configuration in which S has reduced to $!\mathsf{end}$ and T has reduced to a session type other than nil. Rule [c-sync] allows us to find such a path by choosing the appropriate messages exchanged between client and server. In general, one can observe a dicotomy between the rules [c-inp-out] and [c-out-inp] having a universal flavor (they have many premises corresponding to every possible interaction between client and server) and the corule [c-sync] having an existential flavor (it has one premise corresponding to a particular interaction between client and server). This is consistent with the fact that we use rules to express a safety property (which is meant to be *invariant* hence preserved by all the possible interactions) and we use the corule to help us expressing a liveness property. This pattern in the usage of rules and corules is quite common in GISs because of their interpretation and it can also be observed in the GIS for fair termination (see Figure 8.4) and, to some extent, in that for fair subtyping as well (see Section 8.5).

Example 8.4.2. *Consider again* $R_2 = !\mathsf{true}.(?0.!\mathsf{end} + ?\mathbb{N}^+.R_2)$ *from Example 8.4.1 and* $T_2 = ?\mathsf{true}.!\mathbb{N}.T_2 + ?\mathsf{false}.?\mathsf{end}$ *from Example 8.2.1. In order to show that* $R_2 \dashv T_2 \in \mathsf{Gen}[\![\mathcal{C}, \mathcal{C}_{\mathsf{co}}]\!]$ *we have to find a possibly infinite derivation for* $R_2 \dashv T_2$ *using the rules in* $\mathcal{C}$ *as well as finite derivations for all of the judgments occurring in this derivation in* $\mathcal{C} \cup \mathcal{C}_{\mathsf{co}}$.

For the former we have

$$\dfrac{\dfrac{\dfrac{}{!\mathsf{end} \dashv T_2}\,[\textsc{c-success}] \qquad \dfrac{\vdots}{R_2 \dashv T_2}}{?0.!\mathsf{end} + ?\mathbb{N}^+.R_2 \dashv !\mathbb{N}.T_2}\,[\textsc{c-inp-out}]}{R_2 \dashv T_2}\,[\textsc{c-out-inp}]$$

where, in the application of [c-inp-out]*, we have collapsed all of the premises corresponding to the* $\mathbb{N}^+$ *messages into a single premise. Thus, we have proved* $R_2 \dashv T_2 \in \mathsf{CoInd}[\![\mathcal{C}]\!]$*. Note the three judgments occurring in the above derivation tree. The finite derivation*

$$\dfrac{\dfrac{\dfrac{}{!\mathsf{end} \dashv T_2}\,[\textsc{c-success}]}{?0.!\mathsf{end} + ?\mathbb{N}^+.R_2 \dashv !\mathbb{N}.T_2}\,[\textsc{c-sync}]}{R_2 \dashv T_2}\,[\textsc{c-sync}]$$

shows that $R_2 \dashv T_2 \in \mathsf{Ind}[\![\mathcal{C} \cup \mathcal{C}_{\mathsf{co}}]\!]$*. We conclude* $R_2 \dashv T_2 \in \mathsf{Gen}[\![\mathcal{C}, \mathcal{C}_{\mathsf{co}}]\!]$. ⌟

Observe that the corule [c-sync] is at once essential and unsound. For example in Example 8.4.2, without it we would be able to derive the judgment $R_2 \dashv S_2$ despite the fact that R_2 is not fair compliant with S_2 (see

Example 8.4.1). At the same time, if we treated [C-SYNC] as a plain rule, we would be able to derive the judgment $!\mathbb{N}.!\mathsf{end} \dashv ?0.?\mathsf{end}$ despite the reduction $!\mathbb{N}.!\mathsf{end} \# ?0.?\mathsf{end} \rightarrow !\mathsf{end} \# \mathsf{nil}$ since *there exists* an interaction that leads to the successful configuration $!\mathsf{end} \# ?\mathsf{end}$ (if the client sends 0) but none of the others does.

Theorem 8.4.1 (Compliance)**.** *For every $R, T \in \mathbb{S}$, the following properties hold:*

1. *R is compliant with T if and only if $R \dashv T \in \mathsf{CoInd}[\![\mathcal{C}]\!]$;*

2. *R is fairly compliant with T if and only if $R \dashv T \in \mathsf{Gen}[\![\mathcal{C}, \mathcal{C}_{\mathsf{co}}]\!]$.*

Proof sketch. We sketch the proof of item (2), which is the most interesting one. In Section 8.6 we will describe the full proof formalized in Agda. For the "if" part, suppose that $R \dashv T \in \mathsf{Gen}[\![\mathcal{C}, \mathcal{C}_{\mathsf{co}}]\!]$ and consider a reduction $R \# T \Rightarrow R' \# T'$. An induction on the length of this reduction, along with [C-INP-OUT] and [C-OUT-INP], allows us to deduce $R' \dashv T' \in \mathsf{Gen}[\![\mathcal{C}, \mathcal{C}_{\mathsf{co}}]\!]$. Then we have $R' \dashv T' \in \mathsf{Ind}[\![\mathcal{C} \cup \mathcal{C}_{\mathsf{co}}]\!]$ by Definition 1.2.5. An induction on this (well-founded) derivation allows us to find a reduction $R' \# T' \Rightarrow !\mathsf{end} \# T''$ such that $T'' \neq \mathsf{nil}$.

For the left to right implication part we apply the bounded coinduction principle (Proposition 1.2.3). Concerning consistency, we show that whenever R is fairly compliant with T we have that $R \dashv T$ is the conclusion of a rule in Figure 8.5 whose premises are pairs of fairly compliant session types. Indeed, from the hypothesis that R is fairly compliant with T we deduce that there exists a derivation $R \# T \Rightarrow !\mathsf{end} \# T'$ for some $T' \neq \mathsf{nil}$. A case analysis on the shape of R and T allows us to deduce that either $R = !\mathsf{end}$ and $T = T' \neq \mathsf{nil}$, in which case the axiom [C-SUCCESS] applies, or that R and T must be input/output session types with opposite polarities such that the sender has at least one non-nil continuation and whose reducts are still fairly compliant (because fair compliance is preserved by reductions). Then either [C-INP-OUT] or [C-OUT-INP] applies. Concerning boundedness, we do an induction on the reduction $R \# T \Rightarrow !\mathsf{end} \# T'$ to build a well-founded tree made of a suitable number of applications of [C-SYNC] topped by a single application of [C-SUCCESS]. □

8.4.2 Agda Formalization

We now use the Agda library for GIS (see Chapter 7) to formally define the inference system for fair compliance shown in Figure 8.5.

As we did in Section 8.3, the first thing to do is to define the universe U of judgments that we want to derive with the inference system. We can equivalently think of fair compliance as of a binary relation on session types or as a predicate over sessions. We take the second point of view, as it allows us to write more compact code later on.

```
U : Set
U = Session
```

Next, we define two data types to represent the *unique names* with which we identify the rules and corules of the GIS. We use the same labels of Figure 8.5 except for the corule [C-SYNC] which we split into *two* symmetric corules to avoid reasoning on opposite polarities.

```
data RuleNames : Set where
  success inp—out out—inp : RuleNames

data CoRuleNames : Set where
  inp—out out—inp : CoRuleNames
```

Again, we recall that there are two different ways of defining rules and corules, depending on whether these have a finite or a possibly infinite number of premises. Clearly, (co)rules with finitely many premises are just a special case of those with possibly infinite ones, but the GIS library provides some syntactic sugar to specify (co)rules of the former kind in a slightly easier way (see Chapter 7). We use a finite rule to specify [C-SUCCESS].

```
success—rule : FinMetaRule U
success—rule .Ctx = Σ[ Se ∈ Session ] Success Se
success—rule .comp (Se , _) = [] , Se
```

Concerning [C-OUT-INP] and [C-INP-OUT], these rules have a possibly infinite set of premises if 𝕍 is infinite. Therefore, we specify the rules using the most general form allowed by the GIS Agda library.

```
out—inp—rule : MetaRule U
out—inp—rule .Ctx =
        Σ[ (f , _) ∈ Continuation × Continuation ] Witness f
out—inp—rule .Pos ((f , _) , _) = Σ[ x ∈ 𝕍 ] x ∈ dom f
out—inp—rule .prems ((f , g) , _) =
        λ (x , _) → f x .force , g x .force
```

```
out−inp−rule .conclu ((f , g) , _) = out f , inp g

inp−out−rule : MetaRule U
inp−out−rule .Ctx =
        Σ[ (_ , g) ∈ Continuation × Continuation ] Witness g
inp−out−rule .Pos ((_ , g) , _) = Σ[ x ∈ 𝕍 ] x ∈ dom g
inp−out−rule .prems ((f , g) , _) =
        λ (x , _) → f x .force , g x .force
inp−out−rule .conclu ((f , g) , _) = inp f , out g
```

In the above rules, the Pos field, which is the *domain* of the function that *generates* the premises, coincides with that of the continuation function corresponding to the output session type, since we want to specify a fair compliance premise for every message that can be sent. The specification of corules is no different from that of plain rules. As we have anticipated, we split [C-SYNC] into two corules, each having exactly one premise.

```
out−inp−corule : FinMetaRule U
out−inp−corule .Ctx =
        Σ[ (f , _) ∈ Continuation × Continuation ] Witness f
out−inp−corule .comp ((f , g) , x , _) =
        (f x .force , g x .force) :: [] , (out f , inp g)

inp−out−corule : FinMetaRule U
inp−out−corule .Ctx =
        Σ[ (_ , g) ∈ Continuation × Continuation ] Witness g
inp−out−corule .comp ((f , g) , x , _) =
        (f x .force , g x .force) :: [] , (inp f , out g)
```

We can now define two inference systems, FCompIS that consists of the plain rules only and FCompCOIS that consists of the corules only. These are called $\mathcal{C}$ and $\mathcal{C}_{\mathsf{co}}$ in Section 8.4.1.

```
FCompIS : IS U
FCompIS .Names = RuleNames
FCompIS .rules success = from success−rule
FCompIS .rules out−inp = out−inp−rule
FCompIS .rules inp−out = inp−out−rule

FCompCOIS : IS U
FCompCOIS .Names = CoRuleNames
FCompCOIS .rules out−inp = from out−inp−corule
FCompCOIS .rules inp−out = from inp−out−corule
```

where the Agda function `from` converts a finite rule into its more general form on-the-fly, so that the internal representation of all rules is uniform.

We obtain the generalized interpretation of $\langle \mathcal{C}, \mathcal{C}_{\mathsf{co}} \rangle$, named FCompG, through the library function Gen. We also define a predicate FCompI as the inductive interpretation of the union of FCompIS and FCompCOIS, which is useful in the soundness and boundedness proofs of the GIS.

```
FCompG : Session → Set
FCompG = Gen⟦ FCompIS , FCompCOIS ⟧

FCompI : Session → Set
FCompI = Ind⟦ FCompIS ∪ FCompCOIS ⟧
```

The relation ⊣ defined by the GIS in Figure 8.5 is now just a curried version of FCompG.

```
_⊣_ : SessionType → SessionType → Set
R ⊣ S = FCompG (R , S)
```

We conclude this section without showing correctness results as they will be detailed separately in Section 8.6.

8.5 Fair Subtyping

The notions of compliance given in Section 8.4 induce corresponding semantic notions of subtyping that can be used to define a safe substitution principle for session types [Liskov and Wing, 1994]. Intuitively, S is a subtype of T if any client that successfully interacts with S does so with T as well. The reader may look at Chapter 3 for more details about *fair subtyping*.

8.5.1 Definition

As we noted at the beginning, the aim of this chapter is to show how to obtain a refined, liveness enforcing, property by adding corules to the inference systems that alone characterize well known safety properties. Hence, for what concerns subtyping, we refer to the GIS that we presented in Section 3.2.3. However, we have to take into account that the compliance relation the we illustrated in Section 8.4 is *asymmetric*. Thus, we first

$$\overset{\text{S-NIL}}{\frac{}{\mathsf{nil} \leqslant T}} \qquad \overset{\text{S-END}}{\frac{}{\pi\mathsf{end} \leqslant T}\ T \neq \mathsf{nil}}$$

$$\overset{\text{S-INP}}{\frac{\forall x \in X : S_x \leqslant T_x}{?\{x : S_x\}_{x \in X} \leqslant ?\{x : T_x\}_{x \in X \cup Y}}} \qquad \overset{\text{S-OUT}}{\frac{\forall x \in X : S_x \leqslant T_x}{!\{x : S_x\}_{x \in X \cup Y} \leqslant !\{x : T_x\}_{x \in X}}}$$

$$\frac{\forall \varphi \in \mathtt{tr}(S) \setminus \mathtt{tr}(T) : \exists \psi \leq \varphi, x \in \mathbb{V} : S(\psi!x) \leqslant T(\psi!x)}{S \leqslant T}$$

Figure 8.6: Generalized inference system $\langle \mathcal{F}, \mathcal{F}_{\mathsf{co}} \rangle$ for fair subtyping

recall the semantic definitions of (un)fair subtyping and then we show the asymmetric variant of the GIS in Figure 3.4.

Definition 8.5.1 (Subtyping)**.** We say that S is a *subtype* of T if R compliant with S implies R compliant with T for every R.

Definition 8.5.2 (Fair Subtyping)**.** We say that S is a *fair subtype* of T if R fairly compliant with S implies R fairly compliant with T for every R.

The GIS in Figure 8.6 corresponds to that in Figure 3.4 where the two axioms [S-NIL] and [S-END] model the asymmetry. The corule is defined exactly as [FS-CONVERGE] in Figure 3.4. The explanation of the *convergence* is given in Section 3.2.3.

Example 8.5.1. *Consider once again the session types T_i and S_i from Example 8.2.1. The (infinite) derivation*

$$\frac{\dfrac{\dfrac{\vdots}{T_1 \leqslant S_1}}{!\mathbb{N}.T_1 \leqslant !\mathbb{N}^+.S_1}\,[\text{S-OUT}] \qquad \dfrac{}{?\mathsf{end} \leqslant ?\mathsf{end}}\,[\text{S-END}]}{T_1 \leqslant S_1}\,[\text{S-OUT}]$$

proves that $T_1 \leqslant S_1 \in \mathsf{CoInd}[\![\mathcal{F}]\!]$ and the (infinite) derivation

$$\frac{\dfrac{\dfrac{\vdots}{T_2 \leqslant S_2}}{!\mathbb{N}.T_2 \leqslant !\mathbb{N}^+.S_2}\,[\text{S-OUT}] \qquad \dfrac{}{?\mathsf{end} \leqslant ?\mathsf{end}}\,[\text{S-END}]}{T_2 \leqslant S_2}\,[\text{S-INP}]$$

proves that $T_2 \leqslant S_2 \in \mathsf{CoInd}[\![\mathcal{F}]\!]$. In order to derive $T_i \leqslant S_i$ in the GIS $\langle \mathcal{F}, \mathcal{F}_{\mathsf{co}} \rangle$ we must find a well-founded proof tree in $\mathcal{F} \cup \mathcal{F}_{\mathsf{co}}$ and the only hope to do so is by means of [FS-CONVERGE]*, since T_i and S_i share traces of arbitrary length. Observe that every trace φ of T_1 that is not a trace of S_1 has the form $(!\mathsf{true}!p_k)^k!\mathsf{true}!0\dots$ where $p_k \in \mathbb{N}^+$. Thus, it suffices to take $\psi = \varepsilon$ and $x = 0$, noted that $T_1(!0) = S_1(!0) = \pi\mathsf{end}$, to derive*

$$\dfrac{\dfrac{}{?\mathsf{end} \leqslant ?\mathsf{end}}\,[\textsc{fs-converge}]}{T_1 \leqslant S_1}\,[\textsc{fs-converge}]$$

On the other hand, traces $\varphi \in \mathtt{tr}(T_2) \setminus \mathtt{tr}(S_2) = (?\mathsf{true}!p_k)^k?\mathsf{true}!0\dots$ where $p_k \in \mathbb{N}^+$. All the prefixes of such traces that are followed by an output and are shared by both T_2 and S_2 have the form $(?\mathsf{true}!p_k)^k?\mathsf{true}$ where $p_k \in \mathbb{N}^+$, and $T_2(\psi!p) = T_2$ and $S_2(\psi!p) = S_2$ for all such prefixes and $p \in \mathbb{N}^+$. It follows that we are unable to derive $T_2 \leqslant S_2$ with a well-founded proof tree in $\mathcal{F} \cup \mathcal{F}_{\mathsf{co}}$. This is consistent with the fact that, in Example 8.4.1, we have found a client R_2 that is fairly compliant with T_2 but not with S_2. Intuitively, R_2 insists on poking the server waiting to receive 0. This may happen with T_2, but not with S_2. In the case of T_1 and S_1 no such client can exist, since the server may decide to interrupt the interaction at any time by sending a false *message to the client.* ⌟

Remark 8.5.1. Part of the reason why rule [FS-CONVERGE] is so contrived and hard to understand is that the property it enforces is fundamentally *non-local* and therefore difficult to express in terms of immediate subtrees of a session type as we saw for the purely coinductive formulation of fair subtyping (see Section 3.2). To better illustrate the point, consider the following alternative set of corules meant to replace [FS-CONVERGE]:

$$\textsc{co-inc}\quad \dfrac{}{S \leqslant T}\ \mathtt{tr}(S) \subseteq \mathtt{tr}(T) \qquad\qquad \textsc{co-inp}\quad \dfrac{\forall x \in \mathbb{V} S_x \leqslant T_x}{?\{x : S_x\}_{x \in \mathbb{V}} \leqslant ?\{x : T_x\}_{x \in \mathbb{V}}}$$

$$\textsc{co-out}\quad \dfrac{S \leqslant T}{!x.S + S' \leqslant !x.T + T'}$$

It is easy to see that these rules provide a sound approximation of [S-CONVERGE], but they are not complete. Indeed, consider the session types $S = ?\mathsf{true}.S + ?\mathsf{false}.(!\mathsf{true}.?\mathsf{end} + !\mathsf{false}.?\mathsf{end})$ and $T = ?\mathsf{true}.T + ?\mathsf{false}.!\mathsf{true}.?\mathsf{end}$. We have $S \leqslant T$ and yet $S \leqslant_{\mathsf{ind}} T$ cannot be proved with the above corules: it is not possible to prove $S \leqslant_{\mathsf{ind}} T$ using [CO-INC] because $\mathtt{tr}(S) \not\subseteq \mathtt{tr}(T)$. If, on the other hand, we insist on visiting both branches of the topmost input as required by [CO-INP], we end up requiring a proof of $S \leqslant_{\mathsf{ind}} T$ in order to derive $S \leqslant_{\mathsf{ind}} T$. ⌟

Theorem 8.5.1. *For every $S, T \in \mathbb{S}$ the following properties hold:*

1. *S is a subtype of T if and only if $S \leqslant T \in \mathsf{CoInd}[\![\mathcal{F}]\!]$;*
2. *S is a fair subtype of T if and only if $S \leqslant T \in \mathsf{Gen}[\![\mathcal{F}, \mathcal{F}_{\mathsf{co}}]\!]$.*

Proof sketch. As usual we focus on item (2), which is the most interesting property. We have already presented the correctness proofs for the purely coinductive formulation of fair subtyping in Sections 3.2.1 and 3.2.2. So we just sketch the proofs for the GIS one. For the "if" part, we consider an arbitrary R that fairly complies with S and show that it fairly complies with T as well. More specifically, we consider a reduction $R \# T \Rightarrow R' \# T'$ and show that it can be extended so as to achieve client satisfaction. The first step is to "unzip" this reduction into $R \overset{\overline{\varphi}}{\Longrightarrow} R'$ and $T \overset{\varphi}{\Longrightarrow} T'$ for some string φ of actions. Then, we show by induction on φ that there exists S' such that $S' \leqslant T' \in \mathsf{Gen}[\![\mathcal{F}, \mathcal{F}_{\mathsf{co}}]\!]$ and $S \overset{\varphi}{\Longrightarrow} S'$, using the hypothesis $S \leqslant T \in \mathsf{CoInd}[\![\mathcal{F}]\!]$ and the hypothesis that R complies with S. This means that R and S may synchronize just like R and T, obtaining a reduction $R \# S \Rightarrow R' \# S'$. At this point the existence of the reduction $R' \# T' \Rightarrow !\mathsf{end} \# T''$ is proved using the arguments in the discussion of rule [FS-CONVERGE] given earlier.

For the "only if" part we use once again the bounded coinduction principle. In particular, we use the hypothesis that S is a fair subtype of T to show that S and T must have one of the forms in the conclusions of the rules in Figure 8.6. This proof is done by cases on the shape of S, constructing a canonical client of S that must succeed with T as well. Then, the coinduction principle allows us to conclude that $S \leqslant T \in \mathsf{CoInd}[\![\mathcal{F}]\!]$. The fact that $S \leqslant T \in \mathsf{Ind}[\![\mathcal{F} \cup \mathcal{F}_{\mathsf{co}}]\!]$ also holds is by far the most intricate step of the proof. First of all, we establish that $\mathsf{Ind}[\![\mathcal{F} \cup \mathcal{F}_{\mathsf{co}}]\!] = \mathsf{Ind}[\![\mathcal{F}_{\mathsf{co}}]\!]$. That is, we establish that rule [FS-CONVERGE] subsumes all the rules in $\mathcal{F}$ when they are inductively interpreted. Then, we provide a characterization of the *negation* of [FS-CONVERGE], which we call divergence. At this point we proceed by contradiction: under the hypothesis that $S \leqslant T \in \mathsf{CoInd}[\![\mathcal{F}]\!]$ and that S and T "diverge", we are able to corecursively define a *discriminating* (see

Definition 3.2.6) client R that fairly complies with S but not with T. This contradicts the hypothesis that S is a fair subtype of T and proves, albeit in a non-constructive way, that $S \leqslant T \in \mathsf{Ind}[\![\mathcal{F}_{\mathsf{co}}]\!]$ as requested. $\square$

Remark 8.5.2. Most session type theories adopt a symmetric form of session type compatibility whereby client and server are required to terminate the interaction at the same time. It is easy to define a notion of *symmetric compliance* by turning $T' \neq \mathsf{nil}$ into $T' = ?\mathsf{end}$ in Definition 8.4.1. The subtyping relation induced by symmetric compliance has essentially the same characterization of Definition 8.5.1, except that the axiom [S-END] is replaced by the more familiar [FS-END] [Gay and Hole, 2005]. On the other hand, the analogous change in Definition 8.4.2 has much deeper consequences: the requirement that client and server must end the interaction at the same time induces a large family of session types that are syntactically very different, but semantically equivalent. For example, the session types S and T such that $S = ?\mathbb{N}.S$ and $T = !\mathbb{B}.T$, which describe completely unrelated protocols, would be equivalent for the simple reason that no client successfully interacts with them (they are not fairly terminating, since they do not contain any occurrence of end). ⌟

8.5.2 Agda Formalization

Now we show the mechanization of the GIS in Figure 8.6. We still start defining the right universe and then we move to the definition of the (co)rules. At last, we show the key lemmas that are needed to prove the correctness of the predicate. The universe is clearly made of pairs of session types.

```
U : Set
U = SessionType × SessionType
```

Then we have to define names of the rules and the corules. For the sake of clarity, we try to be consistent with the names in Figure 8.6.

```
data FSubIS-RN : Set where
  s-nil s-end : FSubIS-RN
  s-inp s-out : FSubIS-RN

data FSubCOIS-RN : Set where
  converge : FSubCOIS-RN
```

where the corules involve only [FS-CONVERGE]. We start looking at the definitions of the axioms. In this case we can use the finitary formulation by using FinMetaRule .

```
s−nil−r : FinMetaRule U
s−nil−r .Ctx = SessionType
s−nil−r .comp T =
  [] ,
 ─────────────────
  (nil , T)

s−end−r : FinMetaRule U
s−end−r .Ctx =
  Σ[ (S , T) ∈ SessionType × SessionType ] End S × Defined T
s−end−r .comp ((S , T) , _) =
  [] ,
 ─────────────────
  (S , T)
```

Note that the definitions are consistent with [S-NIL] and [S-END]- Indeed s−end−r requires that the first session type has the form πend and that the second is different from nil by using the predicates End and Defined , respectively (see Section 8.2 for more details). Next, we show the rules for modeling covariance of input and contravariance of output.

```
s−inp−r : MetaRule U
s−inp−r .Ctx =
  Σ[ (f , g) ∈ Continuation × Continuation ] dom f ⊆ dom g
s−inp−r .Pos ((f , _) , _) = Σ[ t ∈ 𝕍 ] t ∈ dom f
s−inp−r .prems ((f , g) , _) (t , _) = f t .force , g t .force
s−inp−r .conclu ((f , g) , _) = inp f , inp g

s−out−r : MetaRule U
s−out−r .Ctx = Σ[ (f , g) ∈ Continuation × Continuation ]
                 dom g ⊆ dom f × Witness g
s−out−r .Pos ((_ , g) , _) = Σ[ t ∈ 𝕍 ] t ∈ dom g
s−out−r .prems ((f , g) , _) (t , _) = f t .force , g t .force
s−out−r .conclu ((f , g) , _) = out f , out g
```

We can point out a couple of things. First, both rules have a premise for each element in the domain of the first/second Continuation . Then, co/contra-variance is encoded through the domain inclusion as side condition. We recall that Witness g means that the domain of g is not empty. Now we can move to the [FS-CONVERGE] corule.

```
converge−r : FinMetaRule U
converge−r .Ctx =
  Σ[ (S , T) ∈ SessionType × SessionType ] S ↓ T
converge−r .comp ((S , T) , _) =
  [] ,
  ──────────────────
  (S , T)
```

where S ↓ T is the predicate obtained by inductively encoding the corule [FS-CONVERGE] alone. Although this approach might seem inconsistent with the other mechanizations for it does not use the actual corule, it can be proved that it is equivalent in this example. We will give more details in Section 8.5.3.

For the sake of simplicity, we omit its definition. As usual, we proceed by defining the inference systems.

```
FSubIS : IS U
FSubIS .Names = FSubIS−RN
FSubIS .rules s−nil = from s−nil−r
FSubIS .rules s−end = from s−end−r
FSubIS .rules s−inp = s−inp−r
FSubIS .rules s−out = s−out−r

FSubCOIS : IS U
FSubCOIS .Names = FSubCOIS−RN
FSubCOIS .rules converge = from converge−r
```

At last, we can encode the desired predicate by the generalized interpretation. We also consider the inductive interpretation of the whole inference system which will be used later.

```
_≼F_ : SessionType → SessionType → Set
S ≼F T = FCoInd⟦ FSubIS , FSubCOIS ⟧ (S , T)

_≼Fᵢ_ : SessionType → SessionType → Set
S ≼Fᵢ T = Ind⟦ FSubIS ∪ FSubCOIS ⟧ (S , T)
```

Concerning the specification against which we prove the correctness of _≼F_ it is important to highlight that it can be defined both by using *ground* notions and by _⊣_ (fair compliance) defined in Section 8.4.2. In the mechanization we proved that they are equivalent and we used them differently according to our needs. Here, show only that using the compliance predicate.

```
FSSpec : U → Set
FSSpec (S , T) = ∀{R} → R ⊣ S → R ⊣ T
```

To see the advantage of relying on such specification based on _⊣_, we just need to think that the conclusion R ⊣ T is defined using a GIS. Hence, for what concerns the soundness, we can formulate an ad-hoc specification and use the bounded coinduction principle (Proposition 1.2.3). We omit some details.

```
SpecAux : U → Set
SpecAux (R , T) = Σ[ S ∈ SessionType ] S ⩽F T × R ⊣ S

spec—aux—sound : SpecAux ⊆ λ (R , S) → R ⊣ S
spec—aux—sound =
  bounded—coind[ FCompIS , FCompCOIS ]
    SpecAux spec—bounded spec—cons

sound : ∀{S T} → S ⩽F T → FSSpec (S , T)
sound {S} fs fc = spec—aux—sound (S , fs , fc)
```

where spec − bounded and spec − cons are boundedness and consistency proofs of SpecAux with respect to the GIS in Figure 8.5. For what concerns the completeness, the proof is clearly by bounded coinduction (Proposition 1.2.3).

```
bounded : ∀{S T} → FSSpec (S , T) → S ⩽Fi T

consistent : FSSpec ⊆ ISF[ FSubIS ] FSSpec

complete : ∀{S T} → FSSpec (S , T) → S ⩽F T
complete =
  bounded—coind[ FSubIS , FSubCOIS ] FSSpec bounded consistent
```

8.5.3 On the Corule

We dedicate this last part of the section answering a couple of questions about the corule [FS-CONVERGE] and its mechanization. First, the reader might be confused about whether [FS-CONVERGE] is actually a (meta)rule for it contains an existential quantifier. Hence, [FS-CONVERGE] is not a proper metarule written in that form. Indeed, in the premise both a universal and an existential quantifier are mixed. The issue can be solved by turning such premise in *Skolem normal form*, that is, without existential quantifiers. The drawback of this approach is that quantifiers are replaced by functions making the defined predicate hard to understand. Although for the sake of clarity we could present [FS-CONVERGE] in a *wrong format*, in the Agda mechanization we should have followed the Skolemization approach because the `MetaRule` datatype requires a precise formulation of all the components. For the sake of simplicity, we decided to opt for the more convenient approach that we presented in Section 8.5.2 and that we comment here.

The second question that we try to answer is whether the approach that we adopted in Section 8.5.2 for defining [FS-CONVERGE], that is by using a coaxiom with a side condition, is equivalent to defining the actual corule. Notably, the side condition `S ↓ T` is obtained by considering [FS-CONVERGE] as a stand-alone predicate. Let us show such rule forgetting about subtyping. We use the notation with message tags that we adopted in Section 1.3.

$$\frac{\forall \varphi \in \mathtt{tr}(S) \setminus \mathtt{tr}(T) : \exists \psi \leq \varphi, \mathsf{m} : S(\psi!\mathsf{m}) \downarrow T(\psi!\mathsf{m})}{S \downarrow T}$$

The Agda predicate $\_\downarrow\_$ is defined analogously. Now we can prove that the predicate obtained by inductively interpreting such rule is equivalent to the inductive interpretation of the whole inference system in Figure 8.6.

Lemma 8.5.1. *$S \leqslant_{\mathsf{ind}} T$ if and only if $S \downarrow T$.*

Proof. The “if” part is trivial since the sole rule defining $\downarrow$ is the same as [FS-CONVERGE]. We prove the “only if” part by induction on the derivation of $S \leqslant_{\mathsf{ind}} T$ and by cases on the last rule applied.

Case [S-NIL]. Then $S = \mathsf{nil}$ and $T \neq \mathsf{nil}$. We conclude $S \downarrow T$ observing that $\emptyset \setminus \mathtt{tr}(T) = \emptyset$.

Case [S-END]. Then $S = \pi\mathsf{end}$ for some π and $T \neq \mathsf{nil}$. Notably, $\mathtt{tr}(\pi\mathsf{end}) = \emptyset$ by Definition 8.3.1. Hence, we conclude $S \downarrow T$ observing that $\emptyset \setminus \mathtt{tr}(T) = \emptyset$.

Case [S-INP]. Then $S = ?\{\mathsf{m}_i : S_i\}_{i\in I}$ and $T = ?\{\mathsf{m}_j : T_j\}_{j\in J}$ and $I \subseteq J$ and $S_i \leqslant_{\mathsf{ind}} T_i$ for every $i \in I$. Using the induction hypothesis we deduce that $S_i \downarrow T_i$ for every $i \in I$. We conclude $S \downarrow T$ observing that $\varphi \in \mathtt{tr}(S) \setminus \mathtt{tr}(T)$ implies $\varphi = ?\mathsf{m}_k\psi$ for some $k \in I$ and $\psi \in \mathtt{tr}(S_k) \setminus \mathtt{tr}(T_k)$.

Case [S-OUT]. Then $S = !\{\mathsf{m}_i : S_i\}_{i\in I}$ and $T = !\{\mathsf{m}_j : T_j\}_{j\in J}$ and $J \subseteq I$ and $S_j \leqslant_{\mathsf{ind}} T_j$ for every $j \in J$. Using the induction hypothesis we deduce that $S_j \downarrow T_j$ for every $j \in J$. In order to conclude $S \downarrow T$ we have to show that, for every $\varphi \in \texttt{tr}(S) \setminus \texttt{tr}(T)$, we are able to find $\psi \leq \varphi$ and $\mathsf{m} \in \mathbb{V}$ such that $S(\psi!\mathsf{m}) \downarrow T(\psi!\mathsf{m})$. We distinguish two sub-cases:

- Sub-case $\varphi = !\mathsf{m}_j\varphi'$ where $\varphi' \in \texttt{tr}(S_j) \setminus \texttt{tr}(T_j)$ for some $j \in J$. From $S_j \downarrow T_j$ we deduce that there exist $\psi' \leq \varphi'$ and $\mathsf{m} \in \mathbb{V}$ such that $S_j(\psi'!\mathsf{m}) \downarrow T_j(\psi'!\mathsf{m})$. We conclude by taking $\psi \overset{\mathsf{def}}{=} !\mathsf{m}_j\psi'$ observing that $S(\psi!\mathsf{m}) = S_j(\psi'!\mathsf{m})$ and $T(\psi!\mathsf{m}) = T_j(\psi'!\mathsf{m})$.

- Sub-case $\varphi = !\mathsf{m}_i\varphi'$ where $\varphi' \in \texttt{tr}(S_i)$ for some $i \in I \setminus J$. We conclude by taking $\psi \overset{\mathsf{def}}{=} \varepsilon$ and $\mathsf{m} \overset{\mathsf{def}}{=} \mathsf{m}_j$ for some $j \in J$ and observing that $S(\psi!\mathsf{m}) = S_j$ and $T(\psi!\mathsf{m}) = T_j$.

Case [FS-CONVERGE]. Then, for every $\varphi \in \texttt{tr}(S) \setminus \texttt{tr}(T)$, there exist $\psi \leq \varphi$ and $\mathsf{m} \in \mathbb{V}$ such that $S(\psi!\mathsf{m}) \leqslant_{\mathsf{ind}} T(\psi!\mathsf{m})$. We conclude immediately using the induction hypothesis to deduce that for each such ψ and m we have $S(\psi!\mathsf{m}) \downarrow T(\psi!\mathsf{m})$. $\square$

In the Agda mechanization it happens that we take the inductive interpretation of the GIS with the corule replace by the coaxion. Hence, in order to prove the equivalence between the two approaches, we need to prove that the inductive interpretation of the GIS in Agda is equivalent to the $\downarrow$ predicate.

```
⩽Fᵢ→↓ : ∀{S T} → S ⩽Fᵢ T → S ↓ T
```

We omit the detailed proof (it can be found in Ciccone and Padovani [2021c]). The other direction is trivial since we can directly apply the coaxiom providing the convergence proof.

8.6 Correctness of Fair Compliance

In this section we detail the mechanized correctness proof of *fair compliance* (see point 2. of Theorem 8.4.1). As usual, correctness is expressed in terms of *soundness* and *completeness* of $\_\dashv\_$ with respect to some specification. Such specification is defined in Definition 8.4.2. Before looking at such proofs, we formalize Definition 8.4.2. To formalize fair compliance, we define a `Success` predicate that characterizes those configurations $R \# S$ in which the client has succeeded ($R = !\mathsf{end}$) and the server has not failed ($S \neq \mathsf{nil}$).

```
data Success : Session → Set where
  success : ∀{R S} → Win R → Defined S → Success (R , S)
```

We can weaken Success to MaySucceed , to characterize those configurations *that can be extended* so as to become successful ones. For this purpose we make use of the Satisfiable predicate and of the intersection ∩ of two sets from Agda's standard library.

```
MaySucceed : Session → Set
MaySucceed Se =
        Relation.Unary.Satisfiable (Reductions Se ∩ Success)
```

In words, Se may succeed if there exists Se ' such that Se ⇒ Se' and Se ' is a successful configuration. We can now formulate fair compliance as the property of those sessions that may succeed no matter how they reduce (see Definition 8.4.2). This is the specification against which we prove soundness and completeness of the GIS for fair compliance (Figure 8.5).

```
FCompS : Session → Set
FCompS Se = ∀{Se'} → Reductions Se Se' → MaySucceed Se'
```

Remark 8.6.1. As we did in Section 8.3, assume that we want to instaciate Specification from Section 8.1 by using Session (see Section 8.2.2) as set of states and Reduction (see Section 8.2.3) as transition system. Such instance of Specification would not be equivalent to FCompS as fari compliance corresponds to a successful form of fair termination. In order to make them equivalent we would need to parametrize WeaklyTerminating on a predicate that the reducts have to satisfy. In this case we would say that a state is weakly terminating if it reduces to another one that is in normal form and that satisfies the input predicate. ⌟

8.6.1 Soundness

𝄢 GISs provide no canonical way for proving the soundness of the generalized interpretation of an inference system, so we have to handcraft the proof. We start by proving that the inductive interpretation of the inference system with the corules implies the existence of a reduction leading to a successful configuration.

```
FCompl→MS : ∀{Se} → FCompl Se → MaySucceed Se
FCompl→MS (fold (inj₁ success , (_ , succ) , refl , _)) =
  _ , ϵ , succ
FCompl→MS (fold (inj₁ out−inp , (_ , _ , fx) , refl , pr)) =
  let _ , reds , succ = FCompl→MS (pr (_ , fx)) in
  _ , sync (out fx) inp ◁ reds , succ
FCompl→MS (fold (inj₁ inp−out , (_ , _ , gx) , refl , pr)) =
  let _ , reds , succ = FCompl→MS (pr (_ , gx)) in
  _ , sync inp (out gx) ◁ reds , succ
FCompl→MS (fold (inj₂ out−inp , (_ , _ , fx) , refl , pr)) =
  let _ , reds , succ = FCompl→MS (pr Data.Fin.zero) in
  _ , sync (out fx) inp ◁ reds , succ
FCompl→MS (fold (inj₂ inp−out , (_ , _ , gx) , refl , pr)) =
  let _ , reds , succ = FCompl→MS (pr Data.Fin.zero) in
  _ , sync inp (out gx) ◁ reds , succ
```

where ϵ and red ◁ reds are the constructors of Star . While the former represents the base case (when there are no reductions), the second represents a chain of reductions starting with the single reduction red followed by the reductions reds . There are two things worth noting here. First, in the union of the two inference systems each rule is identified by a name of the form inj ₁ n or inj ₂ n where n is either the name of a rule or of a corule, respectively. Also, we use the function pr to access the premises of a (co)rule by their position. For the plain rules out − inp and inp − out the position is the witness fx or gx that the value exchanged in the synchronization belongs to the domain of the continuation function of the sender. For the corules out − inp and inp − out, we use the position Data . Fin . Zero to access the first and only premise in their list of premises.

The next auxiliary result establishes a "subject reduction" property for fair compliance: if $R \dashv S$ and $R \# S \Rightarrow R' \# S'$, then $R' \dashv S'$. Note that this property is trivial to prove when we consider the specification of fair compliance (see Definition 8.4.2), but here we are referring to the predicate defined by the GIS. The proof consists of a simple induction on the reduction $R \# S \Rightarrow R' \# S'$.

```
sr : ∀{Se Se'} → FCompG Se → Reductions Se Se' → FCompG Se'
sr fc ϵ = fc
sr fc (_ ◁ _) with fc .CoInd⟦_⟧.unfold
sr _ (sync (out fx) inp ◁ _) |
  success , ((_ , success (out e) _) , _) , refl , _ =
  ⊥−elim (e _ fx)
sr _ (sync inp (out gx) ◁ reds) | inp−out , _ , refl , pr =
  sr (pr (_ , gx)) reds
sr _ (sync (out fx) inp ◁ reds) | out—inp , _ , refl , pr =
  sr (pr (_ , fx)) reds
```

Note that we have an absurd case, in which a successful configuration apparently reduces, which we rule out using false elimination (⊥−elim). The soundness proof is a simple combination of the above auxiliary results. We use the library function fcoind −to− ind (see Figure 7.6) to extract an inductive derivation of FCompl Se from a derivation of FCompG Se in the GIS for fair compliance.

```
sound : ∀{Se} → FCompG Se → FCompS Se
sound fc reds = FCompl→MS (fcoind—to—ind (sr fc reds))
```

8.6.2 Completeness

𝄢 For the completeness result we appeal to the bounded coinduction principle of GISs (Proposition 1.2.3, Figure 7.5), which requires us to prove boundedness and consistency of FCompS. Concerning boundedness, we start by computing a proof of FCompl Se for every session Se that may reduce a successful configuration, by induction on the reduction.

```
MS→FCompl : ∀{Se} → MaySucceed Se → FCompl Se
MS→FCompl (_ , reds , succ) = aux reds succ
  where
    aux : ∀{Se Se'} → Reductions Se Se' →
        Success Se' → FCompl Se
    aux ϵ succ = apply—ind (inj1 success) (_ , succ) λ ()
    aux (sync (out fx) inp ◁ red) succ =
      apply—ind (inj2 out—inp)
        (_ , _ , fx) λ{Data.Fin.zero → aux red succ}
    aux (sync inp (out gx) ◁ red) succ =
      apply—ind (inj2 inp—out)
        (_ , _ , gx) λ{Data.Fin.zero → aux red succ}
```

where apply − ind is the function in the GIS library that applies a rule for an inductively defined predicate. Then, boundedness follows by observing that R fairly compliant with S implies the existence of a successful configuration reachable from $R \# S$.

```
bounded : ∀{Se} → FCompS Se → FCompI Se
bounded fc = MS→FCompI (fc ϵ)
```

Showing that FCompS is consistent means showing that every configuration Se that satisfies FCompS is found in the conclusion of a rule in the inference system FCompIS whose premises are all configurations that in turn satisfy FCompS. This follows by a straightforward case analysis on the *first* reduction of Se that leads to a successful configuration.

```
consistent : ∀{Se} → FCompS Se → ISF[ FCompIS ] FCompS Se
consistent fc with fc ϵ
... | _ , ϵ , succ = success , (_ , succ) , refl , λ ()
... | _ , sync (out fx) inp ◁ _ , _ =
out−inp , (_ , _ , fx) , refl ,
  λ (_ , gx) reds → fc (sync (out gx) inp ◁ reds)
... | _ , sync inp (out gx) ◁ _ , _ =
inp−out , (_ , _ , gx) , refl ,
  λ (_ , fx) reds → fc (sync inp (out fx) ◁ reds)
```

We obtain the completeness proof using the bounded coinduction principle, *i.e.* the library function bounded − coind (see Figure 7.5) which applies the principle to the boundedness and consistency proofs.

```
complete : ∀{Se} → FCompS Se → FCompG Se
complete =
  bounded−coind[ FCompIS , FCompCOIS ] FCompS bounded consistent
```

8.7 Related Work

We have shown that generalized inference systems are an effective framework for defining sound and complete proof systems of (some) combined safety and liveness properties of (dependent) session types (Definitions 8.3.2 and 8.4.2), as well as of a liveness-preserving subtyping relation (Definition 8.5.2).

Properties of Session Types. We think that this achievement is more than a coincidence. One of the fundamental results in model checking states that every property can be expressed as the conjunction of a safety property and a liveness property [Alpern and Schneider, 1985, 1987a, Baier and

Katoen, 2008]. The connections between safety and liveness on one side and coinduction and induction on the other make GISs appropriate for characterizing combined safety and liveness properties.

Murgia [2019] studies a wide range of compliance relations for processes and session types, showing that many of them are fixed points of a functional operator, but not necessarily the least or the greatest ones. In particular, he shows that *progress compliance*, which is akin to our compliance (Definition 8.4.1), is a greatest fixed point and that *should-testing compliance*, which is akin to our fair compliance (Definition 8.4.2), is an intermediate fixed point. These results are consistent with Theorem 8.4.1.

Dependent Session Types. The first theories of dependent session types are those of Toninho et al. [2011] and Griffith and Gunter [2013]. These works augment session types with binders, thus allowing for the specification of message predicates. Toninho and Yoshida [2018] present a full calculus combining functions and processes in which the structure of both types and processes may depend on the content of messages. Thiemann and Vasconcelos [2020] propose a full model of functions and processes enabling a simplified form of dependency whereby the structure of types and processes may depend on labels and possibly natural numbers. They introduce a conditional context extension operator that prevents dependencies on linear values.

Zhou [2019] describes the theory and implementation of a refinement session type system where the type of messages can be refined by predicates that specify their properties and relationships.

Formalizations of Session Type Systems. Thiemann [2019] gives the first mechanized proof of a calculus of functions and sessions. His type system distinguishes between types and session types, but only non-dependent pairs are considered. de Muijnck-Hughes et al. [2019] describe an Idris EDSL where dependent types enable reasoning on value dependencies between exchanged messages. Zalakain and Dardha [2020] give another Agda formalization of the linear π-calculus. They focus exclusively on the process layer and only consider channel types, using typing with leftovers [Allais, 2017]. Rouvoet et al. [2020] describe a technique inspired by separation logic to specify and verify in Agda interpreters using linear resources. Among the case studies they discuss is a linearly-typed lambda calculus with primitives for session communications.

Postludium

Let us briefly summarize the two main contents of this thesis. First, we introduced *fair termination* as a desirable property in systems based on communication as it entails many well known properties that are studied in the literature. However, providing a type system for enforcing fair termination is not straightforward as there exist some peculiar scenarios in which the liveness is compromised. Then, due to the increasing interest of research communities in proof assistants we decided to focus on some mechanization aspects. Notably, generalized inference systems have been a transversal topic throughout the thesis as most of the time we relied on them for characterizing the definitions that we needed.

Type Systems On the one hand, in Chapters 4 and 5 we presented a type system for binary/multiparty sessions which involved *fair subtyping*. Then we showed that fair subtyping must be used carefully, that is, finitely many times when it has a strictly positive weight. The main advantage of the type systems under analysis is that they allow for compositional reasoning. On the other hand, in Chapter 6 we presented a linear logic inspired type system for the linear π-calculus. In this case there is no subtyping relation. A peculiar aspect was that we could type both a fair and an unfair process (see Example 6.5.2) by using the same type/formula. Differently with respect to the type systems in Chapters 4 and 5, the enforcement of fair termination is based on a global *validity condition*. At last, we showed that the type systems cannot be compared which leads to interesting research directions. Apart from the technique being used for developing the type systems, we hope that the reader has been convinced that fair termination is a fundamental property for communication-based calculi.

Agda Mechanizations In Chapter 7 we introduce a library for supporting generalized inference systems in Agda. Differently, to the built-in techniques, the library allows to provide *modular* definitions that can be composed without duplicating notions. We then took advantage of such library for characterizing the properties of session types in Chapter 8. The main aim was to show that well known properties could be turned into *fair* ones by adding corules and without changing the main inference systems (see Section 8.5). The application of foundational aspects to session types tried to provide a useful guideline for those researchers involved in the formalization of session type based calculi. Indeed, we also relied on a peculiar definition of session types that aims at simplifying the definition of the predicates on them. We hope to have provided some helpful suggestions for realizing the dream according to which one day all the proofs will be certified.

Future Work

We conclude the thesis by inspecting some possible research directions that we can take in the future. We are grateful to all the anonymous reviewers of the articles that we have submitted during these years as they suggested many interesting topics that we mention in this section.

Fair Termination of Sessions An open question that may have a relevant practical impact is whether the type system in Chapters 4 and 5 remain *sound* in a setting where communications are *asynchronous*. We expect the answer to be positive, as is the case for other synchronous multiparty session types systems [Scalas and Yoshida, 2019], but we have not worked out the details yet. Then, a natural development of these type systems is their application to a real programming environment. We envision two approaches that can be followed to this aim. A bottom-up approach may apply our static analysis technique to a program (in our process calculus) that is extracted from actual code and that captures the code's communication semantics. We expect that suitable annotations may be necessary to identify those branching parts of the code that represent non-deterministic choices in the program. Most typically, these branches will correspond to finite loops or to queries made to the human user of the program that have several different continuations. A top-down approach may provide programmers with a *generative tool* that, starting from a global specification in the form of a *global type* [Honda et al., 2016], produces template code that is "well-typed by design" and that the programmer subsequently instantiates to a specific application. Scribble [Yoshida et al., 2013, Ancona et al., 2016] is an example of such a tool. Interestingly, the usual notion of global type projectability is not sufficient to entail that the session map resulting from a projection is coherent. However, coherence would be guaranteed by requiring that the projected global type is fairly terminating. Finally, we plan to investigate the adaptation of the type system for ensuring the fair termination in the popular actor-based model. This is a drastically different setting in which the order of messages is not as controllable as in the case of sessions. As a consequence, type based analyses require radically different formalisms such as mailbox types [de'Liguoro and Padovani, 2018], for which the study of fair subtyping and of type systems enforcing fair termination is unexplored.

Validity in πLIN One drawback of the type system proposed in Chapter 6 is that establishing whether a quasi-typed process is also well typed requires a global check on the whole typing derivation. This does not happen in the type systems for sessions in Chapters 4 and 5. The need for a global check seems to arise commonly in infinitary proof systems [Dax et al., 2006, Doumane, 2017, Baelde et al., 2016, 2022], so an obvious aspect to investigate is whether the analysis can be localized. A possible source of inspiration for devising a local type system for πLIN might come from the work of Derakhshan and Pfenning [2019]. They propose a compositional technique for dealing with infinitary typing derivations in a session calculus, although their type system is limited to the additive fragment of linear logic.

Fair Subtyping and Linear Logic In Section 6.5 we showed that the type systems in Chapters 4 and 5 and the one in Chapter 6 cannot be compared. In particular, Example 6.5.2 points out a fundamental difference between the two approaches, that is, the presence or absence of fair subtyping. Given the rich literature exploring the connections between linear logic and session types, πLIN and its type system might provide the right framework for investigating the logical foundations of fair subtyping.

Inference Systems in Agda For future work we plan to extend the library that we presented in Chapter 7 in several directions. The first one is to support other interpretations of inference systems, such as the *regular* one [Dagnino, 2020], which is basically coinductive but allows only proof trees with finitely many distinct subtrees. To this end, useful starting points are works on regular terms and streams [Spadotti, 2016, Uustalu and Veltri, 2017] and on finite sets [Firsov and Uustalu, 2015] in dependent type theories. The challenging part is the finiteness constraint, which is not trivial in a type-theoretic setting. A second direction is to implement other proof techniques for (flexible) coinduction, as parametrized coinduction [Hur et al., 2013] and up-to techniques [Pous, 2016, Danielsson, 2018]. Finally, another direction could be the development of a full framework for composition of inference systems, along the lines of seminal work on module systems [Bracha, 1992]. On the more practical side, a further development is to transform the methodology in an automatic translation. That is, a user should be allowed to write an inference system (with corules) in a natural syntax, and the corresponding Agda types should be generated automatically, either by an external tool, or, more interestingly, using *reflection*, recently added in Agda.

Bibliography

Andreas Abel. Type-based termination, inflationary fixed-points, and mixed inductive-coinductive types. In Dale Miller and Zoltán Ésik, editors, *Proceedings 8th Workshop on Fixed Points in Computer Science, FICS 2012*, volume 77 of *EPTCS*, pages 1–11, 2012. doi: 10.4204/EPTCS.77.1.

Andreas Abel and Brigitte Pientka. Well-founded recursion with copatterns and sized types. *Journal of Functional Programming*, 26:e2, 2016. doi: 10.1017/S0956796816000022.

Andreas Abel, Brigitte Pientka, David Thibodeau, and Anton Setzer. Copatterns: programming infinite structures by observations. In Roberto Giacobazzi and Radhia Cousot, editors, *The 40th Annual ACM Symposium on Principles of Programming Languages, POPL'13*, pages 27–38. ACM, 2013. doi: 10.1145/2429069.2429075.

Andreas Abel, Andrea Vezzosi, and Théo Winterhalter. Normalization by evaluation for sized dependent types. *Proceedings of ACM on Programming Languages*, 1(ICFP):33:1–33:30, 2017. doi: 10.1145/3110277.

Samson Abramsky. Proofs as processes. *Theor. Comput. Sci.*, 135(1):5–9, 1994. doi: 10.1016/0304-3975(94)00103-0.

Samson Abramsky, Simon J. Gay, and Rajagopal Nagarajan. Interaction categories and the foundations of typed concurrent programming. In Manfred Broy, editor, *Proceedings of the NATO Advanced Study Institute on Deductive Program Design, Marktoberdorf, Germany*, pages 35–113, 1996.

Peter Aczel. An introduction to inductive definitions. In Jon Barwise, editor, *Handbook of Mathematical Logic*, volume 90 of *Studies in Logic and the Foundations of Mathematics*, pages 739 – 782. Elsevier, 1977. doi: 10.1016/S0049-237X(08)71120-0.

Guillaume Allais. Typing with leftovers - A mechanization of intuitionistic multiplicative-additive linear logic. In Andreas Abel, Fredrik Nordvall Forsberg, and Ambrus Kaposi, editors, *23rd International Conference on Types for Proofs and Programs, TYPES 2017, May 29-June 1, 2017, Budapest, Hungary*, volume 104 of *LIPIcs*, pages 1:1–1:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi: 10.4230/LIPIcs.TYPES.2017.1.

Bowen Alpern and Fred B. Schneider. Defining liveness. *Inf. Process. Lett.*, 21(4):181–185, 1985. doi: 10.1016/0020-0190(85)90056-0.

Bowen Alpern and Fred B. Schneider. Recognizing safety and liveness. *Distributed Comput.*, 2(3):117–126, 1987a. doi: 10.1007/BF01782772.

Bowen Alpern and Fred B. Schneider. Recognizing safety and liveness. *Distributed Comput.*, 2(3):117–126, 1987b. doi: 10.1007/BF01782772.

Thorsten Altenkirch, Neil Ghani, Peter G. Hancock, Conor McBride, and Peter Morris. Indexed containers. *Journal of Functional Programming*, 25, 2015. doi: 10.1017/S095679681500009X.

Davide Ancona, Viviana Bono, Mario Bravetti, Joana Campos, Giuseppe Castagna, Pierre-Malo Deniélou, Simon J. Gay, Nils Gesbert, Elena Giachino, Raymond Hu, Einar Broch Johnsen, Francisco Martins, Viviana Mascardi, Fabrizio Montesi, Rumyana Neykova, Nicholas Ng, Luca Padovani, Vasco T. Vasconcelos, and Nobuko Yoshida. Behavioral types in programming languages. *Found. Trends Program. Lang.*, 3(2-3):95–230, 2016. doi: 10.1561/2500000031.

Davide Ancona, Francesco Dagnino, and Elena Zucca. Reasoning on divergent computations with coaxioms. *Proceedings of ACM on Programming Languages*, 1(OOPSLA):81:1–81:26, 2017a. doi: 10.1145/3133905.

Davide Ancona, Francesco Dagnino, and Elena Zucca. Generalizing inference systems by coaxioms. In Hongseok Yang, editor, *Programming Languages and Systems - 26th European Symposium on Programming, ESOP 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, volume 10201 of *Lecture Notes in Computer Science*, pages 29–55. Springer, 2017b. doi: 10.1007/978-3-662-54434-1_2.

Davide Ancona, Francesco Dagnino, and Elena Zucca. Modeling infinite behaviour by corules. In Todd D. Millstein, editor, *32nd European Conference on Object-Oriented Programming, ECOOP 2018*, volume 109 of *LIPIcs*, pages 21:1–21:31, Dagstuhl, 2018. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik. doi: 10.4230/LIPIcs.ECOOP.2018.21.

Davide Ancona, Francesco Dagnino, Jurriaan Rot, and Elena Zucca. A big step from finite to infinite computations. *Science of Computer Programming*, 197:102492, 2020. ISSN 0167-6423. doi: 10.1016/j.scico.2020.102492.

Krzysztof R. Apt, Nissim Francez, and Shmuel Katz. Appraising fairness in languages for distributed programming. In *Conference Record of the Fourteenth Annual ACM Symposium on Principles of Programming Languages, Munich, Germany, January 21-23, 1987*, pages 189–198. ACM Press, 1987. doi: 10.1145/41625.41642.

David Baelde, Amina Doumane, and Alexis Saurin. Infinitary proof theory: the multiplicative additive case. In Jean-Marc Talbot and Laurent Regnier, editors, *25th EACSL Annual Conference on Computer Science Logic, CSL 2016, August 29 - September 1, 2016, Marseille, France*, volume 62 of *LIPIcs*, pages 42:1–42:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi: 10.4230/LIPIcs.CSL.2016.42.

David Baelde, Amina Doumane, Denis Kuperberg, and Alexis Saurin. Bouncing threads for circular and non-wellfounded proofs. In *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS, 2022*, 2022. URL `https://arxiv.org/abs/2005.08257`.

Christel Baier and Joost-Pieter Katoen. *Principles of model checking.* MIT Press, 2008. ISBN 978-0-262-02649-9.

Gianluigi Bellin and Philip J. Scott. On the pi-calculus and linear logic. *Theor. Comput. Sci.*, 135(1):11–65, 1994. doi: 10.1016/0304-3975(94)00104-9.

Giovanni Bernardi and Matthew Hennessy. Using higher-order contracts to model session types. *Log. Methods Comput. Sci.*, 12(2), 2016. doi: 10.2168/LMCS-12(2:10)2016.

G. Bracha. *The Programming Language JIGSAW: Mixins, Modularity and Multiple Inheritance.* PhD thesis, Department of Comp. Sci., Univ. of Utah, 1992.

Julian C. Bradfield and Colin Stirling. Modal mu-calculi. In Patrick Blackburn, J. F. A. K. van Benthem, and Frank Wolter, editors, *Handbook of Modal Logic*, volume 3 of *Studies in logic and practical reasoning*, pages 721–756. North-Holland, 2007. doi: 10.1016/s1570-2464(07)80015-2.

Mario Bravetti, Julien Lange, and Gianluigi Zavattaro. Fair refinement for asynchronous session types. In Stefan Kiefer and Christine Tasson, editors, *Foundations of Software Science and Computation Structures - 24th International Conference, FOSSACS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings*, volume 12650 of *Lecture Notes in Computer Science*, pages 144–163. Springer, 2021. doi: 10.1007/978-3-030-71995-1_8.

Luís Caires, Frank Pfenning, and Bernardo Toninho. Linear logic propositions as session types. *Math. Struct. Comput. Sci.*, 26(3):367–423, 2016. doi: 10.1017/S0960129514000218.

Marco Carbone and Søren Debois. A graphical approach to progress for structured communication in web services. In Simon Bliudze, Roberto Bruni, Davide Grohmann, and Alexandra Silva, editors, *Proceedings Third Interaction and Concurrency Experience: Guaranteed Interaction, ICE 2010, Amsterdam, The Netherlands, 10th of June 2010*, volume 38 of *EPTCS*, pages 13–27, 2010. doi: 10.4204/EPTCS.38.4. URL `https://doi.org/10.4204/EPTCS.38.4`.

Marco Carbone, Ornela Dardha, and Fabrizio Montesi. Progress as compositional lock-freedom. In Eva Kühn and Rosario Pugliese, editors, *Coordination Models and Languages - 16th IFIP WG 6.1 International Conference, COORDINATION 2014, Held as Part of the 9th International Federated Conferences on Distributed Computing Techniques, DisCoTec 2014, Berlin, Germany, June 3-5, 2014, Proceedings*, volume 8459 of *Lecture Notes in Computer Science*, pages 49–64. Springer, 2014. doi: 10.1007/978-3-662-43376-8_4. URL `https://doi.org/10.1007/978-3-662-43376-8_4`.

Marco Carbone, Sam Lindley, Fabrizio Montesi, Carsten Schürmann, and Philip Wadler. Coherence generalises duality: A logical explanation of multiparty session types. In Josée Desharnais and Radha Jagadeesan, editors, *27th International Conference on Concurrency Theory, CONCUR 2016, August 23-26, 2016, Québec City, Canada*, volume 59 of *LIPIcs*, pages 33:1–33:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi: 10.4230/LIPIcs.CONCUR.2016.33.

Marco Carbone, Fabrizio Montesi, Carsten Schürmann, and Nobuko Yoshida. Multiparty session types as coherence proofs. *Acta Informatica*, 54(3):243–269, 2017. doi: 10.1007/s00236-016-0285-y.

Giuseppe Castagna, Mariangiola Dezani-Ciancaglini, Elena Giachino, and Luca Padovani. Foundations of session types. In António Porto and Francisco Javier López-Fraguas, editors, *Proceedings of the 11th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, September 7-9, 2009, Coimbra, Portugal*, pages 219–230. ACM, 2009. doi: 10.1145/1599410.1599437.

Luca Ciccone. Flexible coinduction in agda. Master's thesis, DIBRIS, Università di Genova, Italy, 2020. URL `https://arxiv.org/abs/2002.06047`.

Luca Ciccone and Luca Padovani. A dependently typed linear π-calculus in agda. In *PPDP'20: 22nd International Symposium on Principles and Practice of Declarative Programming, Bologna, Italy, 9-10 September, 2020*, pages 8:1–8:14. ACM, 2020. doi: 10.1145/3414080.3414109.

Luca Ciccone and Luca Padovani. Inference Systems with Corules for Fair Subtyping and Liveness Properties of Binary Session Types. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming (ICALP'21)*, volume 198 of *LIPIcs*, pages 125:1–125:16, Dagstuhl, Germany, 2021a. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi: 10.4230/LIPIcs.ICALP.2021.125. URL `https://drops.dagstuhl.de/opus/volltexte/2021/14194/pdf/LIPIcs-ICALP-2021-125.pdf`.

Luca Ciccone and Luca Padovani. FairCheck. GitHub repository, 2021b. URL `https://github.com/boystrange/FairCheck`.

Luca Ciccone and Luca Padovani. Fairsubtypingagda. GitHub repository, 2021c. URL `https://github.com/boystrange/FairSubtypingAgda`.

Luca Ciccone and Luca Padovani. Inference systems with corules for combined safety and liveness properties of binary session types. *Log. Methods Comput. Sci.*, 18(3), 2022a. doi: 10.46298/lmcs-18(3:27)2022. URL `https://doi.org/10.46298/lmcs-18(3:27)2022`.

Luca Ciccone and Luca Padovani. An infinitary proof theory of linear logic ensuring fair termination in the linear π-calculus. In Bartek Klin, Slawomir Lasota, and Anca Muscholl, editors, *33rd International Conference on Concurrency Theory, CONCUR 2022, September 12-16, 2022, Warsaw, Poland*, volume 243 of *LIPIcs*, pages 36:1–36:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022b. doi: 10.4230/LIPIcs.CONCUR.2022.36. URL `https://doi.org/10.4230/LIPIcs.CONCUR.2022.36`.

Luca Ciccone and Luca Padovani. Fair termination of binary sessions. *Proc. ACM Program. Lang.*, 6(POPL):1–30, 2022c. doi: 10.1145/3498666. URL `https://doi.org/10.1145/3498666`.

Luca Ciccone, Francesco Dagnino, and Elena Zucca. Flexible coinduction in agda. In Liron Cohen and Cezary Kaliszyk, editors, *12th International Conference on Interactive Theorem Proving, ITP 2021, June 29 to July 1, 2021, Rome, Italy (Virtual Conference)*, volume 193 of *LIPIcs*, pages 13:1–13:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021a. doi: 10.4230/LIPIcs.ITP.2021.13.

Luca Ciccone, Francesco Dagnino, and Elena Zucca. GisLib. GitHub repository, 2021b. URL `https://lcicc.github.io/inference-systems-agda/`.

Luca Ciccone, Francesco Dagnino, and Luca Padovani. Fair termination of multiparty sessions. In Karim Ali and Jan Vitek, editors, *36th European Conference on Object-Oriented Programming, ECOOP 2022, June 6-10, 2022, Berlin, Germany*, volume 222 of *LIPIcs*, pages 26:1–26:26. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi: 10.4230/LIPIcs.ECOOP.2022.26. URL `https://doi.org/10.4230/LIPIcs.ECOOP.2022.26`.

Bruno Courcelle. Fundamental properties of infinite trees. *Theor. Comput. Sci.*, 25:95–169, 1983. doi: 10.1016/0304-3975(83)90059-2.

Francesco Dagnino. Coaxioms: flexible coinductive definitions by inference systems. *Log. Methods Comput. Sci.*, 15(1), 2019. doi: 10.23638/LMCS-15(1:26)2019.

Francesco Dagnino. Foundations of regular coinduction. Technical report, DIBRIS, University of Genova, June 2020. URL `https://arxiv.org/abs/2006.02887`. Submitted for journal publication.

Francesco Dagnino. Foundations of regular coinduction. *Logical Methods in Computer Science*, 17:2:1–2:29, 2021. doi: 10.46298/lmcs-17(4:2)2021. URL `https://lmcs.episciences.org/8539`.

Nils Anders Danielsson. Up-to techniques using sized types. *Proceedings of ACM on Programming Languages*, 2(POPL):43:1–43:28, 2018. doi: 10.1145/3158131.

Ornela Dardha, Elena Giachino, and Davide Sangiorgi. Session types revisited. *Inf. Comput.*, 256:253–286, 2017. doi: 10.1016/j.ic.2017.06.002.

Christian Dax, Martin Hofmann, and Martin Lange. A proof system for the linear time μ-calculus. In S. Arun-Kumar and Naveen Garg, editors, *FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science, 26th International Conference, Kolkata, India, December 13-15, 2006, Proceedings*, volume 4337 of *Lecture Notes in Computer Science*, pages 273–284. Springer, 2006. doi: 10.1007/11944836_26.

Jan de Muijnck-Hughes, Edwin Brady, and Wim Vanderbauwhede. Value-dependent session design in a dependently typed language. In Francisco Martins and Dominic Orchard, editors, *Proceedings Programming Language Approaches to Concurrency- and Communication-cEntric Software, PLACES@ETAPS 2019, Prague, Czech Republic, 7th April 2019*, volume 291 of *EPTCS*, pages 47–59. Open Publishing Association, 2019. doi: 10.4204/EPTCS.291.5.

Ugo de'Liguoro and Luca Padovani. Mailbox types for unordered interactions. In Todd D. Millstein, editor, *32nd European Conference on Object-Oriented Programming, ECOOP 2018, July 16-21, 2018, Amsterdam, The Netherlands*, volume 109 of *LIPIcs*, pages 15:1–15:28. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi: 10.4230/LIPIcs.ECOOP.2018.15.

Farzaneh Derakhshan. *Session-Typed Recursive Processes and Circular Proofs.* PhD thesis, Carnegie Mellon University, 2021. URL `https://www.andrew.cmu.edu/user/fderakhs/publications/Dissertation_Farzaneh.pdf`.

Farzaneh Derakhshan and Frank Pfenning. Circular proofs as session-typed processes: A local validity condition. *CoRR*, abs/1908.01909, 2019. URL `http://arxiv.org/abs/1908.01909`.

Henry DeYoung, Luís Caires, Frank Pfenning, and Bernardo Toninho. Cut reduction in linear logic as asynchronous session-typed communication. In Patrick Cégielski and Arnaud Durand, editors, *Computer Science Logic (CSL'12) - 26th International Workshop/21st Annual Conference of the EACSL, CSL 2012, September 3-6, 2012, Fontainebleau, France*, volume 16 of *LIPIcs*, pages 228–242. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012. doi: 10.4230/LIPIcs.CSL.2012.228.

Amina Doumane. *On the infinitary proof theory of logics with fixed points. (Théorie de la démonstration infinitaire pour les logiques à points fixes)*. PhD thesis, Paris Diderot University, France, 2017. URL `https://tel.archives-ouvertes.fr/tel-01676953`.

Denis Firsov and Tarmo Uustalu. Dependently typed programming with finite sets. In Patrick Bahr and Sebastian Erdweg, editors, *Proceedings of the 11th ACM Workshop on Generic Programming, WGP@ICFP 2015*, pages 33–44. ACM, 2015. doi: 10.1145/2808098.2808102.

Nissim Francez. *Fairness*. Texts and Monographs in Computer Science. Springer, 1986. ISBN 978-3-540-96235-9. doi: 10.1007/978-1-4612-4886-6.

Philippa Gardner, Cosimo Laneve, and Lucian Wischik. Linear forwarders. *Inf. Comput.*, 205(10):1526–1550, 2007. doi: 10.1016/j.ic.2007.01.006.

Simon J. Gay. Subtyping supports safe session substitution. In Sam Lindley, Conor McBride, Philip W. Trinder, and Donald Sannella, editors, *A List of Successes That Can Change the World - Essays Dedicated to Philip Wadler on the Occasion of His 60th Birthday*, volume 9600 of *Lecture Notes in Computer Science*, pages 95–108. Springer, 2016. doi: 10.1007/978-3-319-30936-1_5.

Simon J. Gay and Malcolm Hole. Subtyping for session types in the pi calculus. *Acta Informatica*, 42(2-3):191–225, 2005. doi: 10.1007/s00236-005-0177-z.

Simon J. Gay and Vasco Thudichum Vasconcelos. Linear type theory for asynchronous session types. *J. Funct. Program.*, 20(1):19–50, 2010. doi: 10.1017/S0956796809990268.

Dennis Griffith and Elsa L. Gunter. Liquidpi: Inferrable dependent session types. In Guillaume Brat, Neha Rungta, and Arnaud Venet, editors, *NASA Formal Methods, 5th International Symposium, NFM 2013, Moffett Field, CA, USA, May 14-16, 2013. Proceedings*, volume 7871 of

Lecture Notes in Computer Science, pages 185–197. Springer, 2013. doi: 10.1007/978-3-642-38088-4_13.

Kohei Honda. Types for dyadic interaction. In Eike Best, editor, *CONCUR '93, 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 23-26, 1993, Proceedings*, volume 715 of *Lecture Notes in Computer Science*, pages 509–523. Springer, 1993. doi: 10.1007/3-540-57208-2_35.

Kohei Honda, Vasco Thudichum Vasconcelos, and Makoto Kubo. Language primitives and type discipline for structured communication-based programming. In Chris Hankin, editor, *Programming Languages and Systems - ESOP'98, 7th European Symposium on Programming, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'98, Lisbon, Portugal, March 28 - April 4, 1998, Proceedings*, volume 1381 of *Lecture Notes in Computer Science*, pages 122–138. Springer, 1998. doi: 10.1007/BFb0053567.

Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In George C. Necula and Philip Wadler, editors, *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008, San Francisco, California, USA, January 7-12, 2008*, pages 273–284. ACM, 2008. doi: 10.1145/1328438.1328472.

Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. *J. ACM*, 63(1):9:1–9:67, 2016. doi: 10.1145/2827695.

Chung-Kil Hur, Georg Neis, Derek Dreyer, and Viktor Vafeiadis. The power of parameterization in coinductive proof. In Roberto Giacobazzi and Radhia Cousot, editors, *The 40th Annual ACM Symposium on Principles of Programming Languages, POPL'13*, pages 193–206. ACM, 2013. doi: 10.1145/2429069.2429093.

Hans Hüttel, Ivan Lanese, Vasco T. Vasconcelos, Luís Caires, Marco Carbone, Pierre-Malo Deniélou, Dimitris Mostrous, Luca Padovani, António Ravara, Emilio Tuosto, Hugo Torres Vieira, and Gianluigi Zavattaro. Foundations of session types and behavioural contracts. *ACM Comput. Surv.*, 49(1):3:1–3:36, 2016. doi: 10.1145/2873052.

Jules Jacobs, Stephanie Balzer, and Robbert Krebbers. Connectivity graphs: a method for proving deadlock freedom based on separation logic. *Proc. ACM Program. Lang.*, 6(POPL):1–33, 2022. doi: 10.1145/3498662.

Naoki Kobayashi. A type system for lock-free processes. *Inf. Comput.*, 177 (2):122–159, 2002a. doi: 10.1006/inco.2002.3171.

Naoki Kobayashi. Type systems for concurrent programs. In *10th Anniversary Colloquium of UNU/IIST*, LNCS 2757, pages 439–453. Springer, 2002b. doi: http://dx.doi.org/10.1007/978-3-540-40007-3_26. Extended version at `http://www.kb.ecei.tohoku.ac.jp/~koba/papers/tutorial-type-extended.pdf`.

Naoki Kobayashi. A new type system for deadlock-free processes. In Christel Baier and Holger Hermanns, editors, *CONCUR 2006 - Concurrency Theory, 17th International Conference, CONCUR 2006, Bonn, Germany, August 27-30, 2006, Proceedings*, volume 4137 of *Lecture Notes in Computer Science*, pages 233–247. Springer, 2006. doi: 10.1007/11817949_16.

Naoki Kobayashi and Cosimo Laneve. Deadlock analysis of unbounded process networks. *Inf. Comput.*, 252:48–70, 2017. doi: 10.1016/j.ic.2016.03.004.

Naoki Kobayashi and Davide Sangiorgi. A hybrid type system for lock-freedom of mobile processes. *ACM Trans. Program. Lang. Syst.*, 32(5): 16:1–16:49, 2010. doi: 10.1145/1745312.1745313.

Naoki Kobayashi, Benjamin C. Pierce, and David N. Turner. Linearity and the pi-calculus. *ACM Trans. Program. Lang. Syst.*, 21(5):914–947, 1999. doi: 10.1145/330249.330251.

Wen Kokke, Fabrizio Montesi, and Marco Peressotti. Taking linear logic apart. In Thomas Ehrhard, Maribel Fernández, Valeria de Paiva, and Lorenzo Tortora de Falco, editors, *Proceedings Joint International Workshop on Linearity & Trends in Linear Logic and Applications, Linearity-TLLA@FLoC 2018, Oxford, UK, 7-8 July 2018*, volume 292 of *EPTCS*, pages 90–103, 2018. doi: 10.4204/EPTCS.292.5.

Wen Kokke, Fabrizio Montesi, and Marco Peressotti. Better late than never: a fully-abstract semantics for classical processes. *Proc. ACM Program. Lang.*, 3(POPL):24:1–24:29, 2019. doi: 10.1145/3290337.

Dexter Kozen. Results on the propositional mu-calculus. *Theor. Comput. Sci.*, 27:333–354, 1983. doi: 10.1016/0304-3975(82)90125-6.

Leslie Lamport. Fairness and hyperfairness. *Distributed Comput.*, 13(4): 239–245, 2000. doi: 10.1007/PL00008921.

Xavier Leroy and Hervé Grall. Coinductive big-step operational semantics. *Information and Computation*, 207(2):284–304, 2009. doi: 10.1016/j.ic.2007.12.004.

Sam Lindley and J. Garrett Morris. A semantics for propositions as sessions. In Jan Vitek, editor, *Programming Languages and Systems - 24th European Symposium on Programming, ESOP 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, volume 9032 of *Lecture Notes in Computer Science*, pages 560–584. Springer, 2015. doi: 10.1007/978-3-662-46669-8_23. URL `https://doi.org/10.1007/978-3-662-46669-8_23`.

Sam Lindley and J. Garrett Morris. Talking bananas: structural recursion for session types. In Jacques Garrigue, Gabriele Keller, and Eijiro Sumii, editors, *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming, ICFP 2016, Nara, Japan, September 18-22, 2016*, pages 434–447. ACM, 2016. doi: 10.1145/2951913.2951921.

Barbara Liskov and Jeannette M. Wing. A behavioral notion of subtyping. *ACM Trans. Program. Lang. Syst.*, 16(6):1811–1841, 1994. doi: 10.1145/197320.197383.

Robin Milner. A theory of type polymorphism in programming. *J. Comput. Syst. Sci.*, 17(3):348–375, 1978. doi: 10.1016/0022-0000(78)90014-4. URL `https://doi.org/10.1016/0022-0000(78)90014-4`.

Maurizio Murgia. A note on compliance relations and fixed points. In Massimo Bartoletti, Ludovic Henrio, Anastasia Mavridou, and Alceste Scalas, editors, *Proceedings 12th Interaction and Concurrency Experience, ICE 2019, Copenhagen, Denmark, 20-21 June 2019*, volume 304 of *EPTCS*, pages 38–47, 2019. doi: 10.4204/EPTCS.304.3.

Susan S. Owicki and Leslie Lamport. Proving liveness properties of concurrent programs. *ACM Trans. Program. Lang. Syst.*, 4(3):455–495, 1982. doi: 10.1145/357172.357178.

Luca Padovani. Fair subtyping for open session types. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming - 40th International Colloquium,* *ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part II*, volume 7966 of *Lecture Notes in Computer Science*, pages 373–384. Springer, 2013. doi: 10.1007/978-3-642-39212-2_34.

Luca Padovani. Deadlock and lock freedom in the linear π-calculus. In Thomas A. Henzinger and Dale Miller, editors, *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, pages 72:1–72:10. ACM, 2014. doi: 10.1145/2603088.2603116.

Luca Padovani. Fair subtyping for multi-party session types. *Math. Struct. Comput. Sci.*, 26(3):424–464, 2016. doi: 10.1017/S096012951400022X.

Luca Padovani. A simple library implementation of binary sessions. *J. Funct. Program.*, 27:e4, 2017. doi: 10.1017/S0956796816000289.

Luca Padovani. Fairtermination. GitHub repository, 2022. URL `https://github.com/boystrange/FairTermination`.

Jorge A. Pérez, Luís Caires, Frank Pfenning, and Bernardo Toninho. Linear logical relations for session-based concurrency. In Helmut Seidl, editor, *Programming Languages and Systems - 21st European Symposium on Programming, ESOP 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, volume 7211 of *Lecture Notes in Computer Science*, pages 539–558. Springer, 2012. doi: 10.1007/978-3-642-28869-2_27.

Damien Pous. Coinduction all the way up. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS'16*, pages 307–316. ACM, 2016. doi: 10.1145/2933575.2934564.

Zesen Qian, G. A. Kavvos, and Lars Birkedal. Client-server sessions in linear logic. *Proc. ACM Program. Lang.*, 5(ICFP):1–31, 2021. doi: 10.1145/3473567.

Jean-Pierre Queille and Joseph Sifakis. Fairness and related properties in transition systems - A temporal logic to deal with fairness. *Acta Informatica*, 19:195–220, 1983. doi: 10.1007/BF00265555.

Pedro Rocha and Luís Caires. Propositions-as-types and shared state. *Proc. ACM Program. Lang.*, 5(ICFP):1–30, 2021. doi: 10.1145/3473584.

Arjen Rouvoet, Casper Bach Poulsen, Robbert Krebbers, and Eelco Visser. Intrinsically-typed definitional interpreters for linear, session-typed languages. In Jasmin Blanchette and Catalin Hritcu, editors, *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, New Orleans, LA, USA, January 20-21, 2020*, pages 284–298. ACM, 2020. doi: 10.1145/3372885.3373818.

Davide Sangiorgi. *Introduction to Bisimulation and Coinduction*. Cambridge University Press, 2011. doi: 10.1017/CBO9780511777110.

Alceste Scalas and Nobuko Yoshida. Less is more: multiparty session types revisited. *Proc. ACM Program. Lang.*, 3(POPL):30:1–30:29, 2019. doi: 10.1145/3290343.

Alceste Scalas, Ornela Dardha, Raymond Hu, and Nobuko Yoshida. A linear decomposition of multiparty sessions for safe distributed programming (artifact). *Dagstuhl Artifacts Ser.*, 3(2):03:1–03:2, 2017. doi: 10.4230/DARTS.3.2.3.

Régis Spadotti. *A mechanized theory of regular trees in dependent type theory. (Une théorie mécanisée des arbres réguliers en théorie des types dépendants)*. PhD thesis, Paul Sabatier University, Toulouse, France, 2016. URL `https://tel.archives-ouvertes.fr/tel-01589656`.

Colin Stirling. *Modal and Temporal Properties of Processes*. Texts in Computer Science. Springer, 2001. ISBN 978-1-4419-3153-5. doi: 10.1007/978-1-4757-3550-5.

The Agda Team. *The Agda Reference Manual*, 2022. URL `http://agda.readthedocs.io/en/latest/index.html`.

Peter Thiemann. Intrinsically-typed mechanized semantics for session types. In Ekaterina Komendantskaya, editor, *Proceedings of the 21st International Symposium on Principles and Practice of Programming Languages, PPDP 2019, Porto, Portugal, October 7-9, 2019*, pages 19:1–19:15. ACM, 2019. doi: 10.1145/3354166.3354184.

Peter Thiemann and Vasco T. Vasconcelos. Label-dependent session types. *Proc. ACM Program. Lang.*, 4(POPL):67:1–67:29, 2020. doi: 10.1145/3371135.

Bernardo Toninho and Nobuko Yoshida. Depending on session-typed processes. In Christel Baier and Ugo Dal Lago, editors, *Foundations of*

Software Science and Computation Structures - 21st International Conference, FOSSACS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, volume 10803 of *Lecture Notes in Computer Science*, pages 128–145. Springer, 2018. doi: 10.1007/978-3-319-89366-2_7.

Bernardo Toninho, Luís Caires, and Frank Pfenning. Dependent session types via intuitionistic linear type theory. In Peter Schneider-Kamp and Michael Hanus, editors, *Proceedings of the 13th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, July 20-22, 2011, Odense, Denmark*, pages 161–172. ACM, 2011. doi: 10.1145/2003476.2003499.

Tarmo Uustalu and Niccolò Veltri. Finiteness and rational sequences, constructively. *Journal of Functional Programming*, 27:e13, 2017. doi: 10.1017/S0956796817000041.

Rob van Glabbeek and Peter Höfner. Progress, justness, and fairness. *ACM Comput. Surv.*, 52(4):69:1–69:38, 2019. doi: 10.1145/3329125.

Rob van Glabbeek, Peter Höfner, and Ross Horne. Assuming just enough fairness to make session types complete for lock-freedom. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–13. IEEE, 2021. doi: 10.1109/LICS52264.2021.9470531.

Philip Wadler. Propositions as sessions. *J. Funct. Program.*, 24(2-3):384–418, 2014. doi: 10.1017/S095679681400001X.

Nobuko Yoshida, Raymond Hu, Rumyana Neykova, and Nicholas Ng. The scribble protocol language. In Martín Abadi and Alberto Lluch-Lafuente, editors, *Trustworthy Global Computing - 8th International Symposium, TGC 2013, Buenos Aires, Argentina, August 30-31, 2013, Revised Selected Papers*, volume 8358 of *Lecture Notes in Computer Science*, pages 22–41. Springer, 2013. doi: 10.1007/978-3-319-05119-2_3.

Uma Zalakain and Ornela Dardha. Typing the linear pi calculus in Agda, 2020. URL `https://github.com/umazalakain/typing-linear-pi`.

Uma Zalakain and Ornela Dardha. Co-contextual typing inference for the linear π-calculus in agda (extended abstract). Extended abstract at Workshop on Type-Driven Development (TyDe) 2021, 2021.

Fangyi Zhou. Refinement session types, 2019. URL `https://www.imperial.ac.uk/media/imperial-college/faculty-of-engineering/computing/public/1819-ug-projects/ZhouF-Refinement-Session-Types.pdf`. BSc Dissertation.

Appendix A

Supplement to Chapter 4

A.1 Algorithms

In this section we discuss the ingredients to obtain a type checking algorithm for the type system presented in Section 4.2. In Appendix A.1.1 we show how to compute the *minimum rank of a process* while in Appendix A.1.2 we present a variant of the rules in Figure 4.4 that allows us to obtain a type checking algorithm. Notably, such algorithm has been implemented in Ciccone and Padovani [2021b]. The tool is available on GitHub and contains the codes of most of the examples from this thesis as well as from Ciccone and Padovani [2022c].

A.1.1 Minimum Rank of a Process

In this section we develop a function to compute the *minimum rank* of a process, namely the least quantity that is necessary in order to find a typing derivation for P. In the following we assume that bound names and casts are annotated with session types as well as with the *weight* of the subtyping relation ($\lceil x, m \rceil P$ means that the subtyping being applied has weight m).

Remark A.1.1. The weight of a fair subtyping application is the least solution of the equations in Definition 3.2.4. We did not carry out an analysis about the decidability, so we require that such information is given by the programmer by annotating the process. If the solution of Definition 3.2.4 can be found in a finite amount of time, then the weight can be inferred. ⌟

Moreover, we assume that a non deterministic choice $P \oplus_k Q$ is labeled with a number $m \in \{1, 2\}$ which identifies the path leading to successful termination. The function is defined below.

Definition A.1.1 (Minimum Rank of a Process)**.** The *minimum rank* of a process P, written $\|P\|$, is the least upper bound to the number of casts that P may need to perform and of sessions that P may need to create in order to terminate. Formally, let $\|P\|_{\mathcal{A}}$ be the function inductively defined by the following equations, where $\mathcal{A}$ is a set of process names:

$$\begin{aligned}
\|\mathsf{done}\|_{\mathcal{A}} = \|\mathsf{close}\, x\|_{\mathcal{A}} &= 0 \\
\|\mathsf{wait}\, x.P\|_{\mathcal{A}} &= \|P\|_{\mathcal{A}} \\
\|A\langle\overline{x}\rangle\|_{\mathcal{A}} &= 0 && \text{if } A \in \mathcal{A} \\
\|A\langle\overline{x}\rangle\|_{\mathcal{A}} &= \|P\|_{\mathcal{A}\cup\{A\}} && \text{if } A \notin \mathcal{A} \text{ and } A(\overline{x}) \triangleq P \\
\|x?(y).P\|_{\mathcal{A}} = \|x!y.P\|_{\mathcal{A}} &= \|P\|_{\mathcal{A}} \\
\|\lceil x, m\rceil P\|_{\mathcal{A}} &= m + \|P\|_{\mathcal{A}} \\
\|(x)(P \mid Q)\|_{\mathcal{A}} &= 1 + \|P\|_{\mathcal{A}} + \|Q\|_{\mathcal{A}} \\
\|x\pi\{\mathsf{m}_i : P_i\}_{i\in I}\|_{\mathcal{A}} &= \textstyle\bigsqcup_{i\in I} \|P_i\|_{\mathcal{A}} \\
\|P_1 \oplus_k P_2\|_{\mathcal{A}} &= \|P_k\|_{\mathcal{A}} && \text{if } k \in \{1, 2\}
\end{aligned}$$

We write $\|P\|$ for $\|P\|_{\emptyset}$.

Now we have to show that this function allows us to compute the minimum rank that is necessary in a typing derivation. The first step is to provide a characterization of those processes that play a primary role in computing $\|\cdot\|$. We do so introducing an order relation $\sqsubseteq$ on processes.

Definition A.1.2 (Termination Path)**.** Let $\sqsubseteq$ be the least preorder such that

$$\frac{}{P_k \sqsubseteq P_1 +_k P_2}\; k \in \{1, 2\} \qquad \frac{}{P \sqsubseteq x!y.P} \qquad \frac{}{P \sqsubseteq x?(y).P}$$

$$\frac{}{P_k \sqsubseteq x\pi\{\mathsf{m}_i : P_i\}_{i\in I}}\; k \in I \qquad \frac{A(\overline{x}) \triangleq P}{P \sqsubseteq A\langle\overline{x}\rangle}$$

We say that P is on a *termination path* of Q if $P \sqsubseteq Q$.

Intuitively, $P \sqsubseteq Q$ means that the rank of Q may be affected by the rank of P, because P occurs along a path that leads Q to termination. Hereafter, we often omit the arguments of a process invocation involved in a process order relation and write, for example, $A \sqsubseteq P$ and $P \sqsubseteq A$ instead of $A\langle\overline{x}\rangle \sqsubseteq P$ and $P \sqsubseteq A\langle\overline{x}\rangle$. The notation $A \sqsubseteq P \sqsubseteq A$, shortcut for $A \sqsubseteq P$

and $P \sqsubseteq A$, means that P is found in between two invocations of the same definition A. These "loops" in termination paths are the dangerous places in which no casts (with strictly positive weight) can be performed and no sessions can be created.

Definition A.1.3 (Safe Program)**.** We say that a program $\{A_i(\overline{x_i}) \stackrel{\triangle}{=} P_i\}_{i\in I}$ is *safe* if $A_i \sqsubseteq P \sqsubseteq A_i$ implies that P is not a cast or a session for every P and $i \in I$.

Note that it is straightforward to define an algorithm that checks whether a program is safe, since the number of processes is finite and so is the number of process names.

Now we show that, in a safe program, the rank of a process invocation corresponds to that of its unfolding. This requires some auxiliary results that allow us to express the relationship between the ranks of a process depending on the set $\mathcal{A}$ or process names used. First, we show that the larger the set of process names, the smaller the rank. Intuitively, this is because in Definition A.1.1 every invocation of a process name that occurs in $\mathcal{A}$ results in a null rank.

Lemma A.1.1. If $\mathcal{A} \subseteq \mathcal{B}$, then $\|P\|_{\mathcal{B}} \leq \|P\|_{\mathcal{A}}$.

Proof. By induction on the definition of rank and by cases on the shape of P. We only discuss the base case in which $P = A\langle\overline{x}\rangle$, distinguishing three sub-cases: if $A \in \mathcal{A}$, then we conclude $\|P\|_{\mathcal{B}} = 0 = \|P\|_{\mathcal{A}}$; if $A \in \mathcal{B} \setminus \mathcal{A}$, then we conclude $\|P\|_{\mathcal{B}} = 0 \leq \|P\|_{\mathcal{A}}$; if $A \notin \mathcal{B}$, then we conclude $\|P\|_{\mathcal{B}} = \|Q\|_{\mathcal{B}\cup\{A\}} \leq \|Q\|_{\mathcal{A}\cup\{A\}} = \|P\|_{\mathcal{A}}$ using the induction hypothesis and $A(\overline{x}) \stackrel{\triangle}{=} Q$. ☐

Next we show that the rank of a process P does not depend on the presence or absence of A in the set $\mathcal{A}$ if $A \not\sqsubseteq P$ if there is no invocation to A along any termination path of P.

Lemma A.1.2. If $A \not\sqsubseteq P$, then $\|P\|_{\mathcal{A}} = \|P\|_{\mathcal{A}\cup\{A\}}$.

Proof. By induction on the definition of $\|P\|_{\mathcal{A}}$ and by cases on the shape of P.

Cases $P = \mathsf{done}$ and $P = \mathsf{close}\, x$. We conclude $\|P\|_{\mathcal{A}} = \|P\|_{\mathcal{A}\cup\{A\}} = 0$.

Case $P = B\langle\overline{x}\rangle$ where $B(\overline{x}) \stackrel{\triangle}{=} Q$. From the hypothesis $A \not\sqsubseteq P$ we deduce $B \neq A$. We distinguish two sub-cases. If $B \in \mathcal{A}$, then we conclude

$$\begin{array}{rcll}
\|P\|_{\mathcal{A}} & = & \|B\langle\overline{x}\rangle\|_{\mathcal{A}} & \text{by definition of } P \\
& = & 0 & \text{by definition of } \|\cdot\| \\
& = & \|B\langle\overline{x}\rangle\|_{\mathcal{A}\cup\{A\}} & \text{by definition of } \|\cdot\| \\
& = & \|P\|_{\mathcal{A}\cup\{A\}} & \text{by definition of } P
\end{array}$$

If $B \notin \mathcal{A}$, then from the hypothesis $A \not\sqsubseteq P$ we deduce $A \not\sqsubseteq Q$ and we conclude

$$\begin{array}{rcll} \|P\|_{\mathcal{A}} & = & \|B\langle\overline{x}\rangle\|_{\mathcal{A}} & \text{by definition of } P \\ & = & \|Q\|_{\mathcal{A}\cup\{B\}} & \text{by definition of } \|\cdot\| \\ & = & \|Q\|_{\mathcal{A}\cup\{A,B\}} & \text{using the induction hypothesis} \\ & = & \|B\langle\overline{x}\rangle\|_{\mathcal{A}\cup\{A\}} & \text{by definition of } \|\cdot\| \\ & = & \|P\|_{\mathcal{A}\cup\{A\}} & \text{by definition of } P \end{array}$$

Case $P = P_1 \oplus_k P_2$ where $k \in \{1,2\}$. From the hypothesis $A \not\sqsubseteq P$ we deduce $A \not\sqsubseteq P_k$. We conclude

$$\begin{array}{rcll} \|P\|_{\mathcal{A}} & = & \|P_1 \oplus_k P_2\|_{\mathcal{A}} & \text{by definition of } P \\ & = & \|P_k\|_{\mathcal{A}} & \text{by definition of } \|\cdot\| \\ & = & \|P_k\|_{\mathcal{A}\cup\{A\}} & \text{using the induction hypothesis} \\ & = & \|P_1 \oplus_k P_2\|_{\mathcal{A}\cup\{A\}} & \text{by definition of } \|\cdot\| \\ & = & \|P\|_{\mathcal{A}\cup\{A\}} & \text{by definition of } P \end{array}$$

Case $P = x!y.Q$. From the hypothesis $A \not\sqsubseteq P$ we deduce $A \not\sqsubseteq Q$. We conclude

$$\begin{array}{rcll} \|P\|_{\mathcal{A}} & = & \|x!y.Q\|_{\mathcal{A}} & \text{by definition of } P \\ & = & \|Q\|_{\mathcal{A}} & \text{by definition of } \|\cdot\| \\ & = & \|Q\|_{\mathcal{A}\cup\{A\}} & \text{using the induction hypothesis} \\ & = & \|x!y.Q\|_{\mathcal{A}\cup\{A\}} & \text{by definition of } \|\cdot\| \\ & = & \|P\|_{\mathcal{A}\cup\{A\}} & \text{by definition of } P \end{array}$$

Case $P = x?(y).Q$. Analogous to the previous case.
Case $P = \lceil x, m\rceil Q$. From $A \not\sqsubseteq P$ we deduce $A \not\sqsubseteq Q$. We conclude

$$\begin{array}{rcll} \|P\|_{\mathcal{A}} & = & \|\lceil x\rceil Q\|_{\mathcal{A}} & \text{by definition of } P \\ & = & m + \|Q\|_{\mathcal{A}} & \text{by definition of } \|\cdot\| \\ & = & m + \|Q\|_{\mathcal{A}\cup\{A\}} & \text{using the induction hypothesis} \\ & = & \|\lceil x\rceil Q\|_{\mathcal{A}\cup\{A\}} & \text{by definition of } \|\cdot\| \\ & = & \|P\|_{\mathcal{A}\cup\{A\}} & \text{by definition of } P \end{array}$$

Case $P = (x)(P_1 \mid P_2)$. From $A \not\sqsubseteq P$ we deduce $A \not\sqsubseteq P_i$ for $i = 1,2$. We conclude

$$\begin{array}{rcll} \|P\|_{\mathcal{A}} & = & \|(x)(P_1 \mid P_2)\|_{\mathcal{A}} & \text{by definition of } P \\ & = & 1 + \|P_1\|_{\mathcal{A}} + \|P_2\|_{\mathcal{A}} & \text{by definition of } \|\cdot\| \\ & = & 1 + \|P_1\|_{\mathcal{A}\cup\{A\}} + \|P_2\|_{\mathcal{A}\cup\{A\}} & \text{using the induction hypothesis} \\ & = & \|(x)(P_1 \mid P_2)\|_{\mathcal{A}\cup\{A\}} & \text{by definition of } \|\cdot\| \\ & = & \|P\|_{\mathcal{A}\cup\{A\}} & \text{by definition of } P \end{array}$$

Case $P = x\pi\{\mathsf{m}_i : P_i\}_{i\in I}$. From the hypothesis $A \not\sqsubseteq P$ we deduce $A \not\sqsubseteq P_i$ for every $i \in I$. We conclude

$$\begin{array}{rcll}
\|P\|_{\mathcal{A}} & = & \|x\pi\{\mathsf{m}_i : P_i\}_{i\in I}\|_{\mathcal{A}} & \text{by definition of } P \\
& = & \bigsqcup_{i\in I} \|P_i\|_{\mathcal{A}} & \text{by definition of } \|\cdot\| \\
& = & \bigsqcup_{i\in I} \|P_i\|_{\mathcal{A}\cup\{A\}} & \text{using the induction hypothesis} \\
& = & \|x\pi\{\mathsf{m}_i : P_i\}_{i\in I}\|_{\mathcal{A}\cup\{A\}} & \text{by definition of } \|\cdot\| \\
& = & \|P\|_{\mathcal{A}\cup\{A\}} & \text{by definition of } P
\end{array}$$

□

Finally, we can show that the rank of a process P such that $A \sqsubseteq P \sqsubseteq A$ cannot exceed the rank of A. Recall that $A \sqsubseteq P$ means that there is an invocation to A along a termination path of P and that $P \sqsubseteq A$ means that P occurs along a termination path of A.

Lemma A.1.3. In a safe program, if $A \sqsubseteq P \sqsubseteq A$ and $A(\overline{x}) \stackrel{\triangle}{=} Q$, then $\|P\|_{\mathcal{A}} \leq \|P\|_{\mathcal{A}\cup\{A\}} \sqcup \|A\langle\overline{x}\rangle\|$.

Proof. By induction on $\|P\|_{\mathcal{A}}$ and by cases on the shape of P. Note that P cannot be a cast or a session because of the hypothesis that the program is safe.

Case $P = B\langle\overline{x}\rangle$ *and* $B \in \mathcal{A}$. We conclude

$$\begin{array}{rcll}
\|P\|_{\mathcal{A}} & = & \|B\langle\overline{x}\rangle\|_{\mathcal{A}} & \text{by definition of } P \\
& = & 0 & \text{by definition of } \|\cdot\| \\
& = & \|B\langle\overline{x}\rangle\|_{\mathcal{A}\cup\{A\}} & \text{by definition of } \|\cdot\| \\
& \leq & \|B\langle\overline{x}\rangle\|_{\mathcal{A}\cup\{A\}} \sqcup \|A\langle\overline{x}\rangle\| & \text{property of } \sqcup \\
& = & \|P\|_{\mathcal{A}\cup\{A\}} \sqcup \|A\langle\overline{x}\rangle\| & \text{by definition of } P
\end{array}$$

Case $P = A\langle\overline{y}\rangle$ *and* $A \notin \mathcal{A}$. We may assume, without loss of generality, that $\overline{y} = \overline{x}$ since the rank of a process invocation does *not* depend on channel names. We conclude

$$\begin{array}{rcll}
\|P\|_{\mathcal{A}} & = & \|A\langle\overline{x}\rangle\|_{\mathcal{A}} & \text{by definition of } P \\
& = & \|Q\|_{\mathcal{A}\cup\{A\}} & \text{by definition of } \|\cdot\| \\
& \leq & \|Q\|_{\{A\}} & \text{by Lemma A.1.1} \\
& = & \|A\langle\overline{x}\rangle\| & \text{by definition of } \|\cdot\| \\
& = & 0 \sqcup \|A\langle\overline{x}\rangle\| & \text{property of } \sqcup \\
& = & \|A\langle\overline{x}\rangle\|_{\mathcal{A}\cup\{A\}} \sqcup \|A\langle\overline{x}\rangle\| & \text{by definition of } \|\cdot\| \\
& = & \|P\|_{\mathcal{A}\cup\{A\}} \sqcup \|A\langle\overline{x}\rangle\| & \text{by definition of } P
\end{array}$$

Case $P = B\langle\overline{x}\rangle$ and $B \notin \mathcal{A} \cup \{A\}$ where $B(\overline{x}) \stackrel{\triangle}{=} R$. From the hypotheses $A \sqsubseteq P \sqsubseteq A$ and $B \notin \mathcal{A} \cup \{A\}$ we deduce $A \sqsubseteq R \sqsubseteq A$.

$$\begin{array}{rcll}
\|P\|_{\mathcal{A}} & = & \|B\langle\overline{x}\rangle\|_{\mathcal{A}} & \text{by definition of } P \\
& = & \|R\|_{\mathcal{A}\cup\{B\}} & \text{by definition of } \|\cdot\| \\
& \leq & \|R\|_{\mathcal{A}\cup\{A,B\}} \sqcup \|A\langle\overline{x}\rangle\| & \text{using the induction hypothesis} \\
& = & \|B\langle\overline{x}\rangle\|_{\mathcal{A}\cup\{A\}} \sqcup \|A\langle\overline{x}\rangle\| & \text{by definition of } \|\cdot\| \\
& & & \text{and the hypothesis } B \notin \mathcal{A} \cup \{A\} \\
& = & \|P\|_{\mathcal{A}\cup\{A\}} \sqcup \|A\langle\overline{x}\rangle\| & \text{by definition of } P
\end{array}$$

Case $P = P_1 \oplus_k P_2$ where $k \in \{1,2\}$. From the hypotheses $A \sqsubseteq P \sqsubseteq A$ we deduce $A \sqsubseteq P_k \sqsubseteq A$. We conclude

$$\begin{array}{rcll}
\|P\|_{\mathcal{A}} & = & \|P_1 +_k P_2\|_{\mathcal{A}} & \text{by definition of } P \\
& = & \|P_k\|_{\mathcal{A}} & \text{by definition of } \|\cdot\| \\
& \leq & \|P_k\|_{\mathcal{A}\cup\{A\}} \sqcup \|A\langle\overline{x}\rangle\| & \text{using the induction hypothesis} \\
& = & \|P_1 \oplus_k P_2\|_{\mathcal{A}\cup\{A\}} \sqcup \|A\langle\overline{x}\rangle\| & \text{by definition of } \|\cdot\| \\
& = & \|P\|_{\mathcal{A}\cup\{A\}} \sqcup \|A\langle\overline{x}\rangle\| & \text{by definition of } P
\end{array}$$

Case $P = x!y.Q$. From the hypotheses $A \sqsubseteq P \sqsubseteq A$ we deduce $A \sqsubseteq Q \sqsubseteq A$. We conclude

$$\begin{array}{rcll}
\|P\|_{\mathcal{A}} & = & \|x!y.Q\|_{\mathcal{A}} & \text{by definition of } P \\
& = & \|Q\|_{\mathcal{A}} & \text{by definition of } \|\cdot\| \\
& \leq & \|Q\|_{\mathcal{A}\cup\{A\}} \sqcup \|A\langle\overline{x}\rangle\| & \text{using the induction hypothesis} \\
& = & \|x!y.Q\|_{\mathcal{A}\cup\{A\}} \sqcup \|A\langle\overline{x}\rangle\| & \text{by definition of } \|\cdot\| \\
& = & \|P\|_{\mathcal{A}\cup\{A\}} \sqcup \|A\langle\overline{x}\rangle\| & \text{by definition of } P
\end{array}$$

Case $P = x?(y).Q$. Analogous to the previous case.

Case $P = x\pi\{\mathsf{m}_i : P_i\}_{i\in I}$. For every $i \in I$ we distinguish two possibilities, depending on whether $A \not\sqsubseteq P_i$ or $A \sqsubseteq P_i$. In the first case we deduce $\|P_i\|_{\mathcal{A}} = \|P_i\|_{\mathcal{A}\cup\{A\}}$ using Lemma A.1.2. In the second case we deduce $\|P_i\|_{\mathcal{A}} \leq \|P_i\|_{\mathcal{A}\cup\{A\}} \sqcup \|A\langle\overline{x}\rangle\|$ using the induction hypothesis. Either way, we have $\|P_i\|_{\mathcal{A}} \leq \|P_i\|_{\mathcal{A}\cup\{A\}} \sqcup \|A\langle\overline{x}\rangle\|$ for every $i \in I$. We conclude

$$\begin{array}{rcll}
\|P\|_{\mathcal{A}} & = & \|x\pi\{\mathsf{m}_i : P_i\}_{i\in I}\|_{\mathcal{A}} & \text{by definition of } P \\
& = & \bigsqcup_{i\in I} \|P_i\|_{\mathcal{A}} & \text{by definition of } \|\cdot\| \\
& \leq & \bigsqcup_{i\in I}(\|P_i\|_{\mathcal{A}\cup\{A\}} \sqcup \|A\langle\overline{x}\rangle\|) & \text{using Lemma A.1.2} \\
& & & \text{or the induction hypothesis} \\
& = & (\bigsqcup_{i\in I} \|P_i\|_{\mathcal{A}\cup\{A\}}) \sqcup \|A\langle\overline{x}\rangle\| & \text{distributivity of } \sqcup \\
& = & \|x\pi\{\mathsf{m}_i : P_i\}_{i\in I}\|_{\mathcal{A}\cup\{A\}} \sqcup \|A\langle\overline{x}\rangle\| & \text{by definition of } \|\cdot\| \\
& = & \|P\|_{\mathcal{A}\cup\{A\}} \sqcup \|A\langle\overline{x}\rangle\| & \text{by definition of } P
\end{array}$$

□

Next is the key lemma stating that the given notion of rank (Definition A.1.1) behaves well, in the sense that it is preserved by unfolding a process invocation.

Lemma A.1.4. In a safe program such that $A(\overline{x}) \stackrel{\triangle}{=} P$ we have $\|P\| = \|A\langle\overline{x}\rangle\|$.

Proof. Clearly $P \sqsubseteq A$ since P is the body of the definition of process A. We distinguish two sub-cases that cover all possibilities. If $A \not\sqsubseteq P$, then using Lemma A.1.2 we deduce $\|P\| = \|P\|_\emptyset = \|P\|_{\{A\}} = \|A\langle\overline{x}\rangle\|$. If $A \sqsubseteq P$, then using Lemma A.1.2 we deduce $\|P\| = \|P\|_\emptyset \leq \|P\|_{\{A\}} \sqcup \|A\langle\overline{x}\rangle\| = \|A\langle\overline{x}\rangle\| \sqcup \|A\langle\overline{x}\rangle\| = \|A\langle\overline{x}\rangle\|$. Using Lemma A.1.1 we conclude $\|A\langle\overline{x}\rangle\| = \|P\|_{\{A\}} \leq \|P\|$. □

In order to show that $\|P\|$ is indeed the minimum rank of P that allows us to find a typing derivation for P, provided there is one, the first thing to do is to prove that a well-typed program is also safe. So from now on, when we reason about well-typed processes, we may assume that they belong to a safe program.

Lemma A.1.5. A well-typed program is safe.

Proof. First of all observe that $\Gamma \vdash^m P$ and $\Delta \vdash^n Q$ and $P \sqsubseteq Q$ imply $m \leq n$. This follows by considering the base cases of $P \sqsubseteq Q$ and looking at the typing rules in which Q is in the conclusion and P is one of the premises. Then, $A \sqsubseteq P \sqsubseteq A$ implies that P occurs in a typing judgment having exactly the same rank annotation as that of A. It follows that P cannot be a cast or a session, for these forms strictly increase the rank annotation. □

Next we show that $\|P\|$ is no greater than any rank that may appear is a typing derivation for P.

Lemma A.1.6. If $\Gamma \vdash^n P$, then $\|P\| \leq n$.

Proof. We prove that $\Gamma \vdash^n P$ implies $\|P\|_{\mathcal{A}} \leq n$ by a straightforward induction on the definition of $\|P\|_{\mathcal{A}}$. The conclusion $\|P\| \leq n$ is then just the particular case when $\mathcal{A} = \emptyset$. □

Finally, we show that if a program is well typed under *some* assignment then it is also well typed under the assignment that uses the minimum ranks. With Lemma A.1.6, this result justifies the definition of $\|P\|$ as *minimum rank* of P.

Theorem A.1.1. *If $\{A_i(\overline{x_i}) \stackrel{\triangle}{=} P_i\}_{i\in I}$ is well typed under the global assignment $\{A_i : [\overline{S_i}; n_i]\}_{i\in I}$, then it is well typed also under the global assignment $\{A_i : [\overline{S_i}; \|P_i\|]\}_{i\in I}$.*

Proof. First we use the coinduction principle to show that every judgment in $\mathcal{R} \stackrel{\mathsf{def}}{=} \{\Gamma \vdash^m P \mid \Gamma \vdash^n_{\mathsf{coind}} P, \|P\| \leq m\}$ is the conclusion of a rule in Figure 4.4 whose premises are also in $\mathcal{R}$. This allows us to deduce that $\Gamma \vdash^n_{\mathsf{coind}} P$ implies $\Gamma \vdash^{\|P\|}_{\mathsf{coind}} P$. Suppose $\Gamma \vdash^m P \in \mathcal{R}$. Then $\Gamma \vdash^n_{\mathsf{coind}} P$ and $\|P\| \leq m$. We reason by cases on the last rule used in the derivation of $\Gamma \vdash^n_{\mathsf{coind}} P$. We only discuss two representative cases.

Case [TB-CALL]. Then $P = A_k\langle\overline{x}\rangle$ and $A_k(\overline{x}) \stackrel{\triangle}{=} Q$ and $\Gamma \vdash^{n'}_{\mathsf{coind}} Q$ for some $n' \leq n$ and some $k \in I$. From the definition of rank we deduce $\|P\| = \|Q\| = \|Q_k\|$ since the rank of a process does not depend on its free names. From Lemma A.1.6 we deduce $\|Q_k\| \leq n_k$. We conclude by observing that $m \geq n_k$ and that $\Gamma \vdash^m Q$ is the conclusion of [TB-CALL].

Case [TB-TAG]. Then $P = x\pi\{\mathsf{m}_i : Q_i\}_{i\in I}$ and $\Gamma = \Delta, x : \pi\{\mathsf{m}_i : S_i\}_{i\in K}$ and $\Delta, x : S_i \vdash^n_{\mathsf{coind}} Q_i$ for every $i \in I$. From the definition of rank we have $\|P\| = \bigsqcup_{i\in I} \|Q_i\|$. From $m \geq \|P\|$ we deduce $m \geq \|Q_i\|$ for every $i \in I$. Then $\Delta, x : S_i \vdash^m Q_i \in \mathcal{R}$ for every $i \in I$ by definition of $\mathcal{R}$ and we conclude by observing that $\Gamma \vdash^m P$ is the conclusion of [TB-LABEL].

To show that $\Gamma \vdash^n_{\mathsf{ind}} P$ implies $\Gamma \vdash^{\|P\|}_{\mathsf{ind}} P$ it suffices a straightforward induction on the derivation of $\Gamma \vdash^n_{\mathsf{ind}} P$. By the bounded coinduction principle this is enough to conclude. $\square$

A.1.2 Type Checking Algorithm

𝄢 In this section we show how to obtain an alternative version of the typing rules from which it is easy to derive a type checking algorithm, provided that bound names and casts are explicitly annotated with session types and the weight of the subtyping being applied. As we pointed out in Appendix A.1.1, we assume that non deterministic choices are labeled with the branch which leads to successful termination. There are three aspects that make the type system presented in Figure 4.4 not strictly algorithmic:

1. the fact that typing derivations are potentially infinite

2. the need for building finite derivations using the corules [COB-CHOICE] and [COB-TAG], which overlap with [TB-CHOICE] and [TB-TAG] respectively

3. the rank annotation to be used in each typing judgment

$$\textsc{a-done}\quad \frac{}{\emptyset \vdash \mathsf{done}} \qquad \textsc{a-call}\quad \frac{}{\overline{x : S} \vdash A\langle\overline{x}\rangle}\; A : [\overline{S}; n] \qquad \textsc{a-choice}\quad \frac{\Gamma \vdash P \qquad \Gamma \vdash Q}{\Gamma \vdash P \oplus Q}$$

$$\textsc{a-close}\quad \frac{}{x : !\mathsf{end} \vdash \mathsf{close}\, x} \qquad \textsc{a-wait}\quad \frac{\Gamma \vdash P}{\Gamma, x : ?\mathsf{end} \vdash \mathsf{wait}\, x.P} \qquad \textsc{a-channel-in}\quad \frac{\Gamma, x : S, y : T \vdash P}{\Gamma, x : ?T.S \vdash x?(y).P}$$

$$\textsc{a-tag}\quad \frac{\forall i \in I : \Gamma, x : S_i \vdash P_i}{\Gamma, x : \pi\{\mathsf{m}_i : S_i\}_{i\in I} \vdash x\pi\{\mathsf{m}_i : P_i\}_{i\in I}} \qquad \textsc{a-channel-out}\quad \frac{\Gamma, x : S \vdash P}{\Gamma, x : !T.S, y : T \vdash x!y.P}$$

$$\textsc{a-par}\quad \frac{\Gamma, x : S \vdash P \qquad \Delta, x : T \vdash Q}{\Gamma, \Delta \vdash (x)(P \mid Q)}\; S \sim T \qquad \textsc{a-cast}\quad \frac{\Gamma, x : T \vdash P}{\Gamma, x : S \vdash \lceil x \rceil P}\; S \leqslant T$$

Figure A.1: Algorithmic typing rules for processes

Concerning Item 3, in Appendix A.1.1 we have seen how the rank annotation can be computed for any process in a safe program. So here we focus on Items 1 and 2.

Figure A.1 presents an (inductively interpreted) set of typing rules that are a "stripped down" version of those given in Figure 4.4. There are two main differences between these rules and those given in the main body of the paper: first, there is no rank annotation in typing judgments; second, [A-CALL] is an *axiom*, unlike [TB-CALL]. The remaining structure of the rules and the kind of constraints they impose is exactly the same as before. Henceforth, we write $\Gamma \vdash_{\mathsf{alg}} P$ if $\Gamma \vdash P$ is (inductively) derivable using the typing rules in Figure A.1.

Lemma A.1.7. Let $\{A_i(\overline{x_i}) \triangleq P_i\}_{i\in I}$ be a safe program and let $\{A_i : [\overline{S_i}; n_i]\}_{i\in I}$ be a global assignment. The following properties are equivalent:

1. $\overline{x_i : S_i} \vdash^{n_i}_{\mathsf{coind}} P_i$ for every $i \in I$;
2. $\overline{x_i : S_i} \vdash_{\mathsf{alg}} P_i$ for every $i \in I$.

Proof. 1 $\Rightarrow$ 2. Just observe that a (finite) derivation for $\Gamma \vdash_{\mathsf{alg}} P$ can be obtained from a (possibly infinite) derivation for $\Gamma \vdash_{\mathsf{coind}} P$ by truncating each application of [TB-CALL] in the latter derivation to an application of [A-CALL] with the same conclusion.

$$
\frac{}{\mathcal{A} \Vdash \mathsf{done}}
\qquad
\frac{}{\mathcal{A} \Vdash \mathsf{close}\, x}
\qquad
\frac{\mathcal{A} \cup \{A\} \Vdash P}{\mathcal{A} \Vdash A\langle \overline{x} \rangle}\; A \notin \mathcal{A}, A(\overline{x}) \stackrel{\triangle}{=} P
$$

$$
\frac{\mathcal{A} \Vdash P_k}{\mathcal{A} \Vdash P_1 \oplus_k P_2}\; k \in \{1,2\}
\qquad
\frac{\mathcal{A} \Vdash P}{\mathcal{A} \Vdash x?(y).P}
\qquad
\frac{\mathcal{A} \Vdash P}{\mathcal{A} \Vdash x!y.P}
$$

$$
\frac{\mathcal{A} \Vdash P_k}{\mathcal{A} \Vdash x\pi\{\mathsf{m}_i : P_i\}_{i\in I}}\; k \in I
\qquad
\frac{\forall i \in \{1,2\} : \mathcal{A} \Vdash P_i}{\mathcal{A} \Vdash (x)(P_1 \mid P_2)}
\qquad
\frac{\mathcal{A} \Vdash P}{\mathcal{A} \Vdash \lceil x \rceil P}
$$

Figure A.2: Algorithmic rules for action boundedness

$2 \Rightarrow 1$. Let $\mathcal{R} \stackrel{\mathsf{def}}{=} \{\Gamma \vdash^n P \mid \Gamma \vdash_{\mathsf{alg}} P, \|P\| \leq n\}$. Using the coinduction principle it suffices to show that each judgment found in $\mathcal{R}$ is the conclusion of a rule in Figure 4.4 whose premises are also in $\mathcal{R}$. Let $\Gamma \vdash^n P \in \mathcal{R}$, meaning that $\Gamma \vdash_{\mathsf{alg}} P$ and $\|P\| \leq n$. We reason by cases on the rule used to derive $\Gamma \vdash_{\mathsf{alg}} P$. We only discuss a few cases.

Case [A-DONE]. We conclude observing that P is the conclusion of [TB-DONE].

Case [A-CALL]. Then $P = A\langle \overline{x} \rangle$ for some $A(\overline{x}) \stackrel{\triangle}{=} Q$. Note that $n \geq \|P\| = \|Q\|$. From the hypothesis we know that $\Gamma \vdash_{\mathsf{alg}} Q$ and we conclude by observing that $\Gamma \vdash^n P$ is the conclusion of [TB-CALL] and that $\Gamma \vdash^n Q \in \mathcal{R}$ by definition of $\mathcal{R}$.

Case [A-PAR]. Then $P = (x)(P_1 \mid P_2)$ and $\Gamma = \Gamma_1, \Gamma_2$ and $\Gamma_i, x : S_i \vdash_{\mathsf{alg}} P_i$ for $i = 1, 2$ and $S_1 \sim S_2$. Note that $n \geq \|P\| = 1 + \|P_1\| + \|P_2\|$. Hence, there exist n_1 and n_2 such that $n = 1 + n_1 + n_2$ and $\|P_i\| \leq n_i$ for $i = 1, 2$. We conclude by observing that $\Gamma \vdash^n P$ is the conclusion of [TB-PAR] and that $\Gamma_i, x : S_i \vdash^{n_i} P_i \in \mathcal{R}$ by definition of $\mathcal{R}$.

Case [A-TAG]. Then $P = x\pi\{\mathsf{m}_i : P_i\}_{i\in I}$ and $\Gamma = \Gamma', x : \pi\{\mathsf{m}_i : S_i\}_{i\in I}$ and $\Gamma', x : S_i \vdash_{\mathsf{alg}} P_i$ for every $i \in I$. Note that $n \geq \|P\| = \bigsqcup_{i\in I} \|P_i\|$, hence $n \geq \|P_i\|$ for every $i \in I$. We conclude by observing that $\Gamma \vdash^n P$ is the conclusion of [TB-TAG] and that $\Gamma', x : S_i \vdash^n P_i \in \mathcal{R}$ by definition of $\mathcal{R}$. $\square$

To filter out those judgments derivable in the algorithmic type system for which there is no finite derivation using the original type system with the corules [COB-CHOICE] and [COB-LABEL], we separately define the (inductive) inference system shown in Figure A.2 for action boundedness. Note that this inference system can be trivially turned into an algorithm by checking whether, for a process of the form $x\pi\{\mathsf{m}_i : P_i\}_{i\in I}$, there is at least one branch for which $\mathcal{A} \Vdash P_i$ is derivable.

Lemma A.1.8. If $\Gamma \vdash^{n}_{\mathsf{coind}} P$, then $\Gamma \vdash^{n}_{\mathsf{ind}} P$ if and only if $\emptyset \Vdash P$.

Proof. For the "if" part we prove that $\mathcal{A} \Vdash P$ implies $\Gamma \vdash^{A}_{\mathsf{ind}} P$ by induction on the derivation of $\mathcal{A} \Vdash P$. For the "only if", we first prove that if $\Gamma \vdash^{A}_{\mathsf{ind}} P$ and none of the process names occurring in the [TB-CALL] applications of this derivation is in $\mathcal{A}$, then $\mathcal{A} \Vdash P$. Then, the result follows by considering the *smallest* derivation $\Gamma \vdash^{A}_{\mathsf{ind}} P$, in which no process definition is expanded twice. $\square$

Theorem A.1.2. *Let $\{A_i(\overline{x_i}) \triangleq P_i\}_{i\in I}$ be a safe program and let $\{A_i : [\overline{S_i}; n_i]\}_{i\in I}$ be a global assignment. The following properties are equivalent:*

1. $\overline{x_i : S_i} \vdash^{n_i} P_i$ *for every* $i \in I$*;*
2. $\overline{x_i : S_i} \vdash_{\mathsf{alg}} P_i$ *for every* $i \in I$ *and* $\emptyset \Vdash Q$ *is derivable for every* Q *occurring in the derivations.*

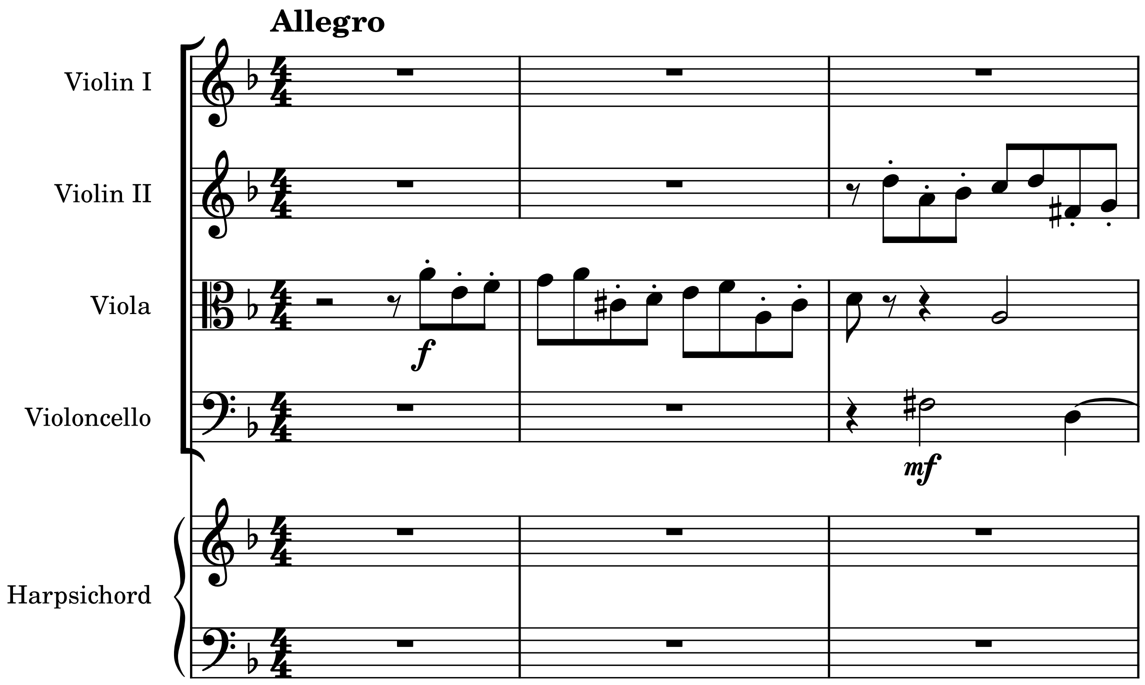
Proof. Consequence of Lemmas A.1.7 and A.1.8. $\square$

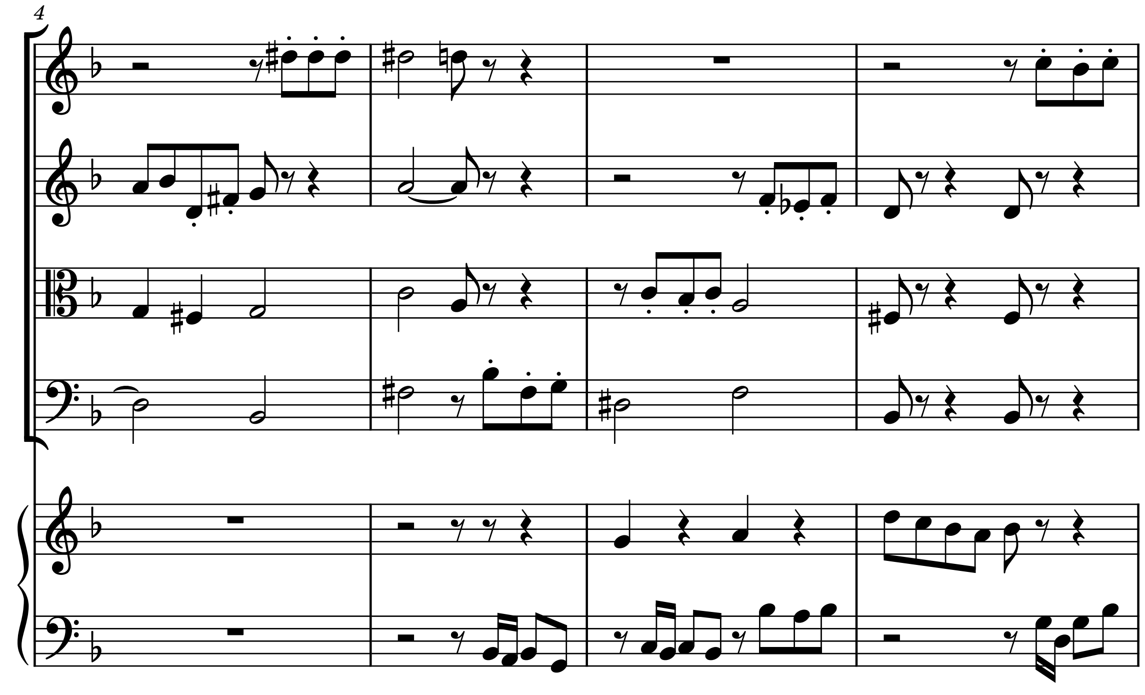
Concertino in Dm

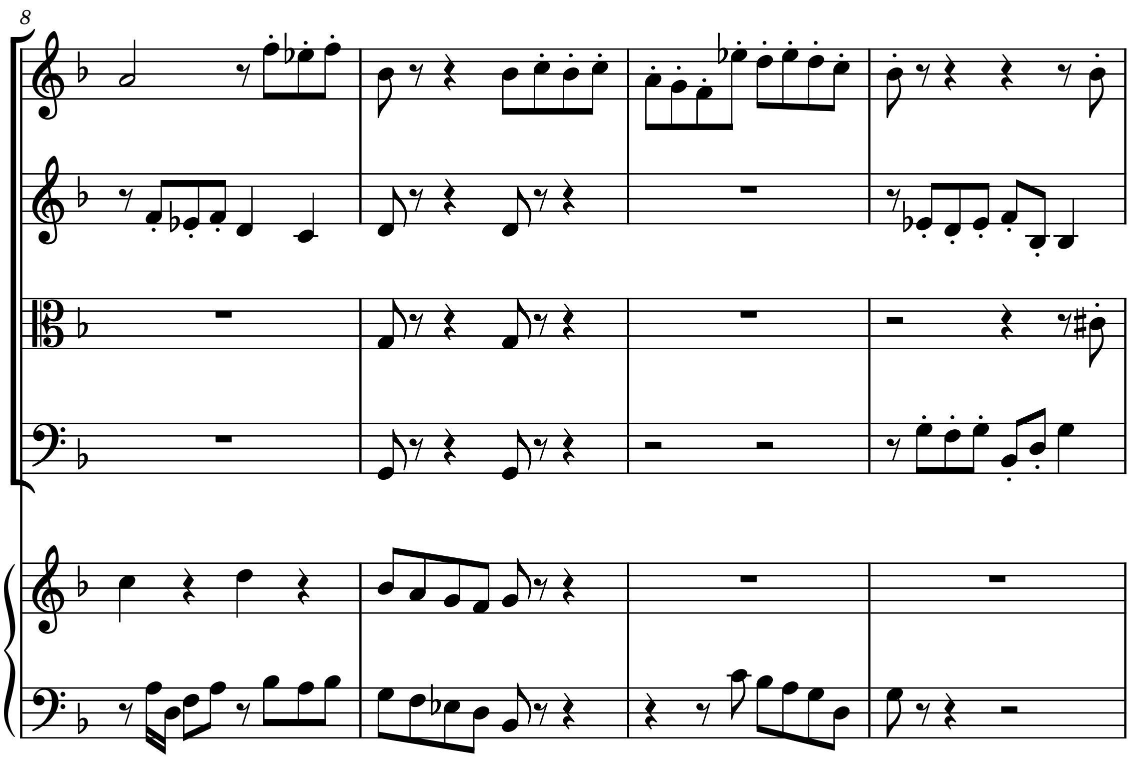
For a PhD

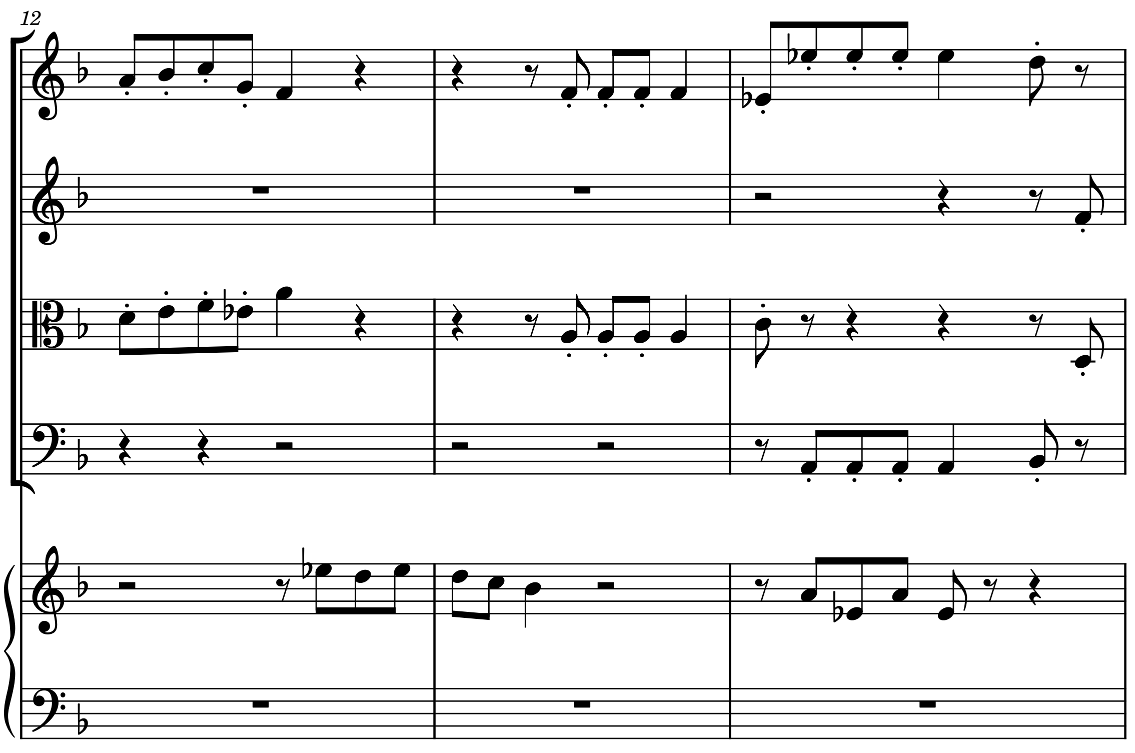
https://musescore.com/user/34410061/scores/9124791

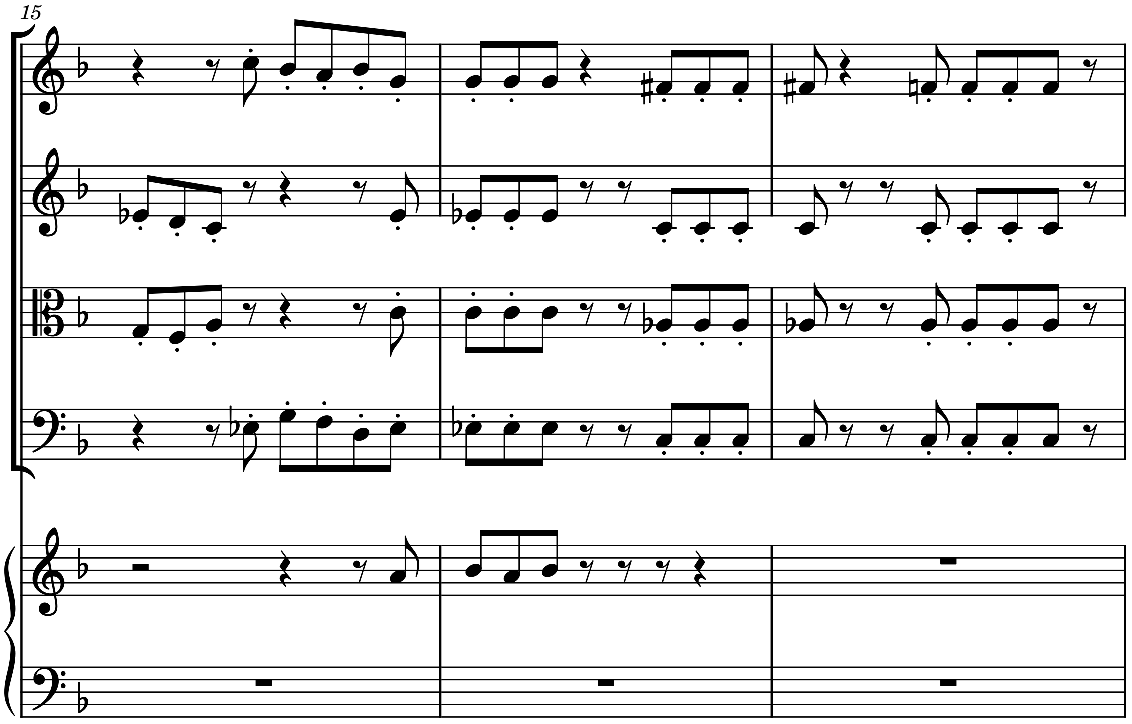
Luca Ciccone

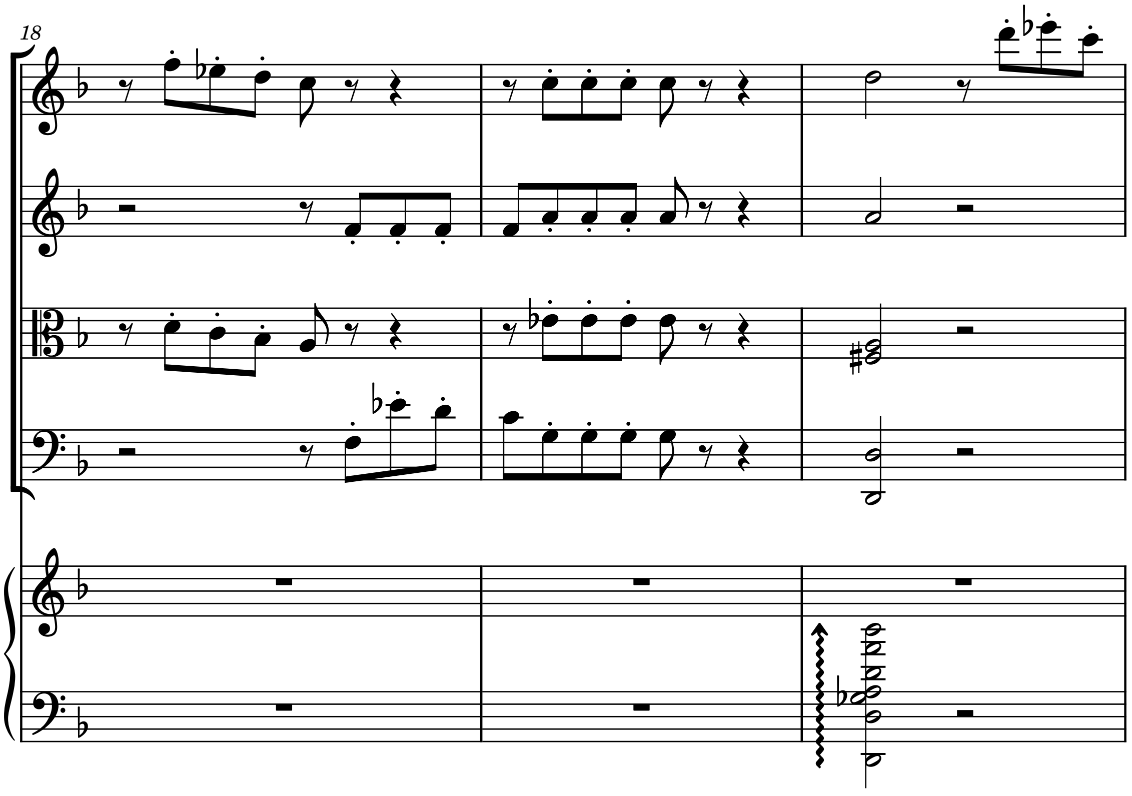


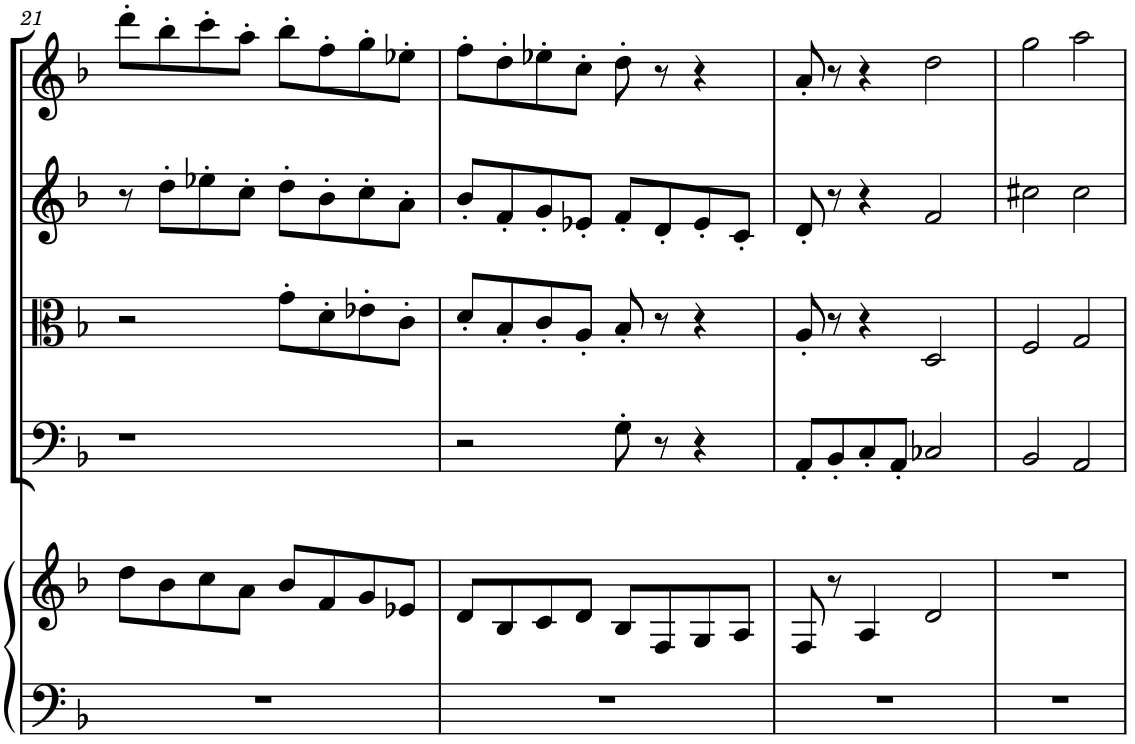


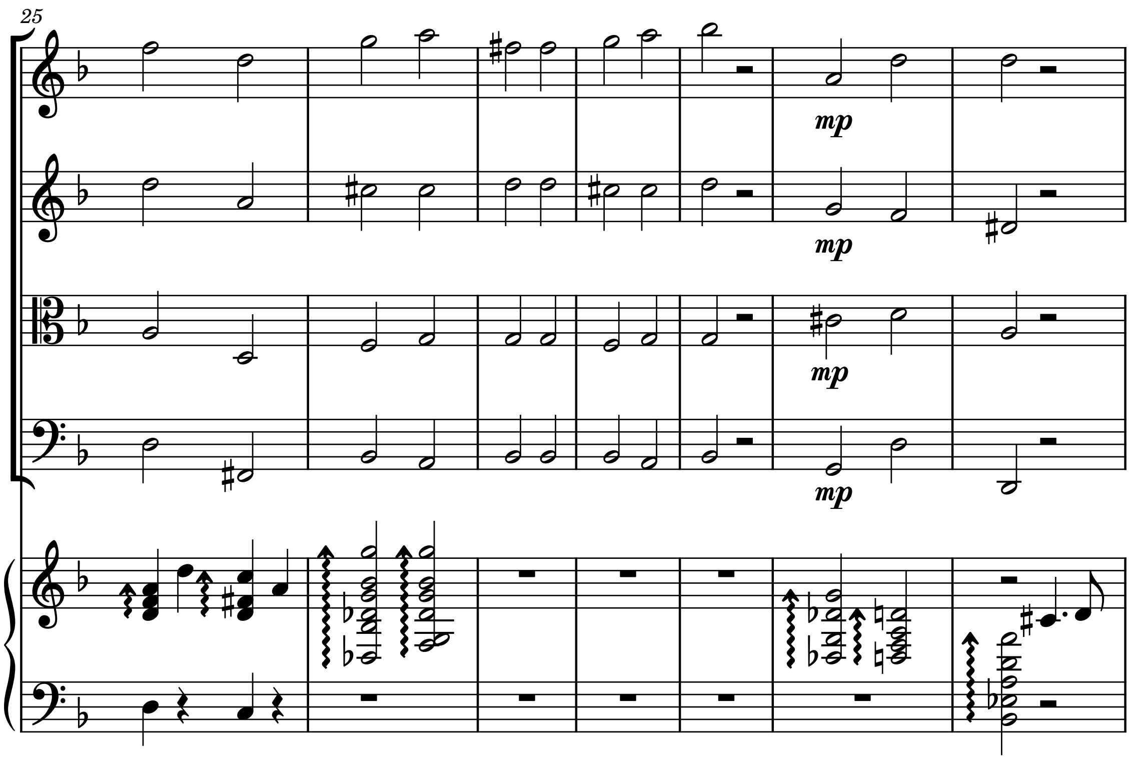

8

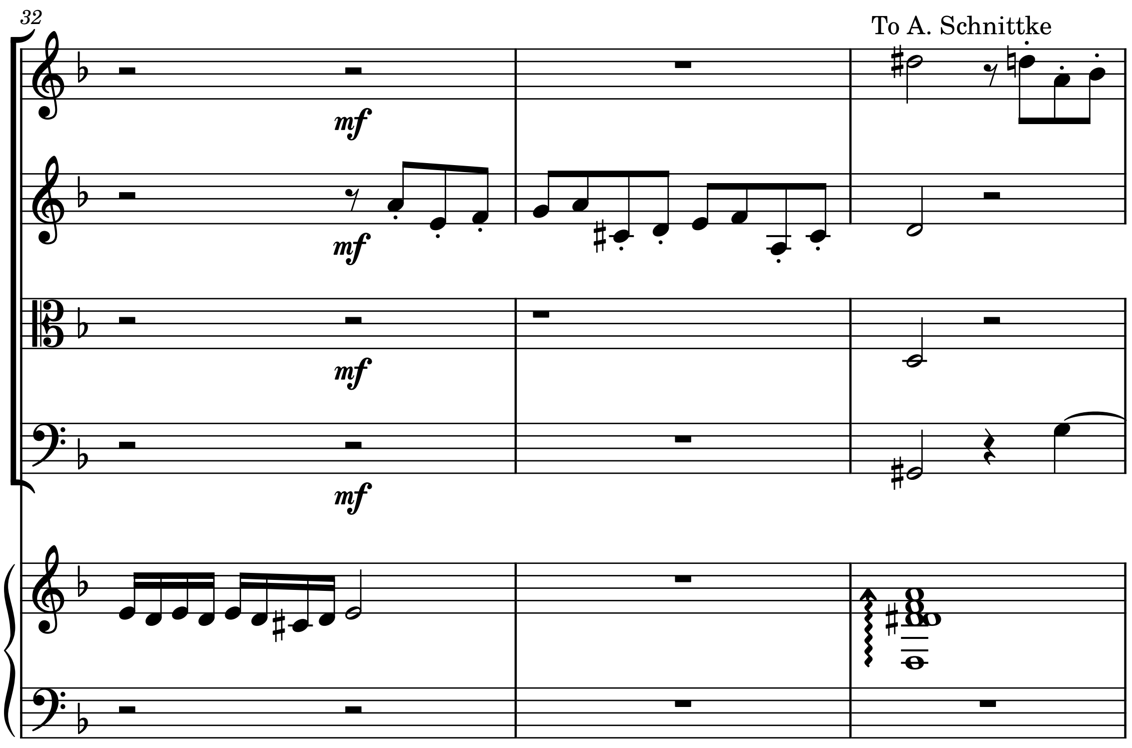

12

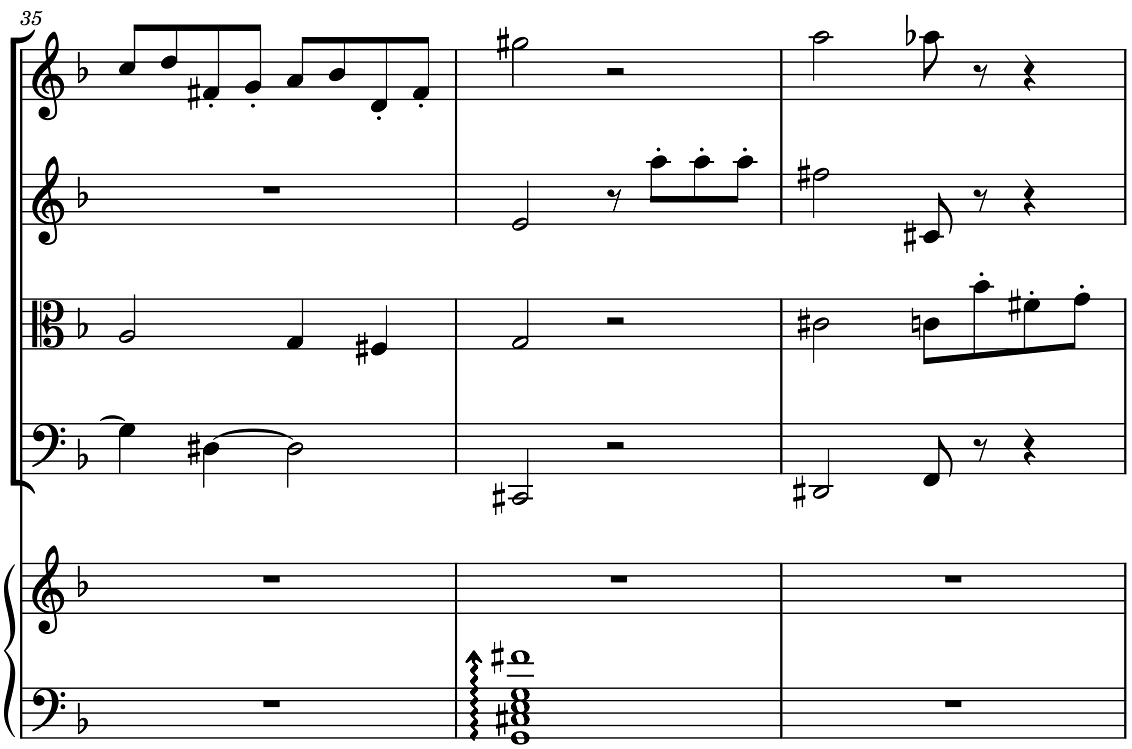

15

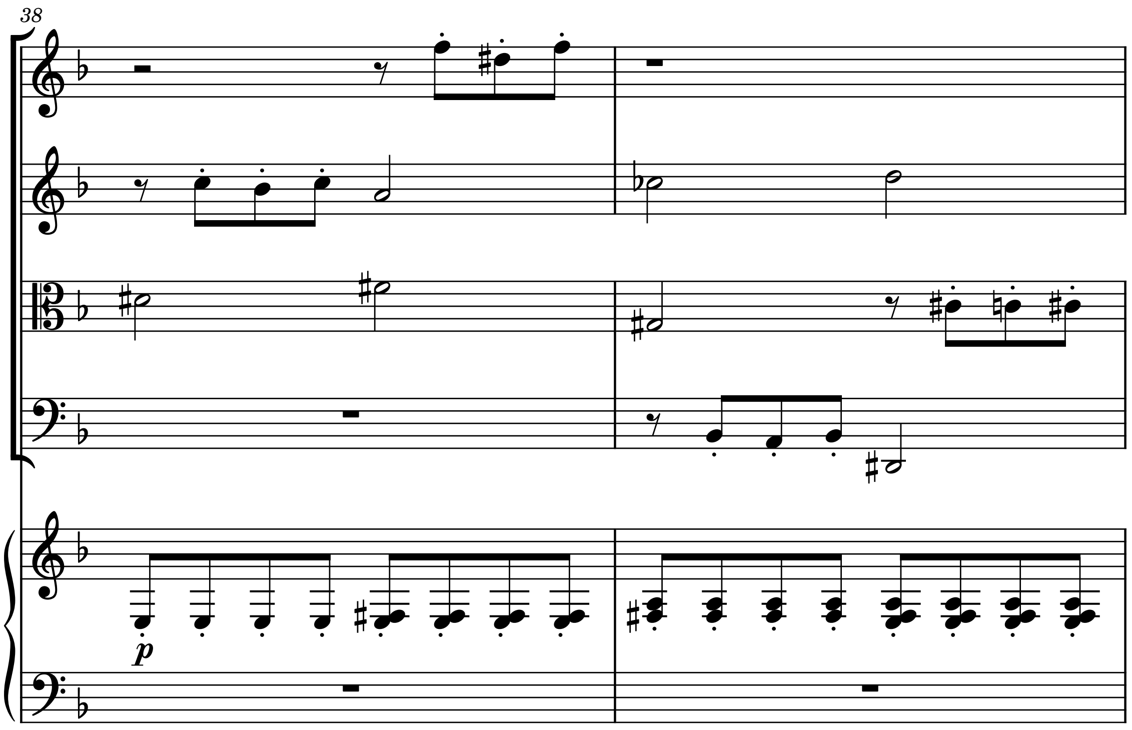

18

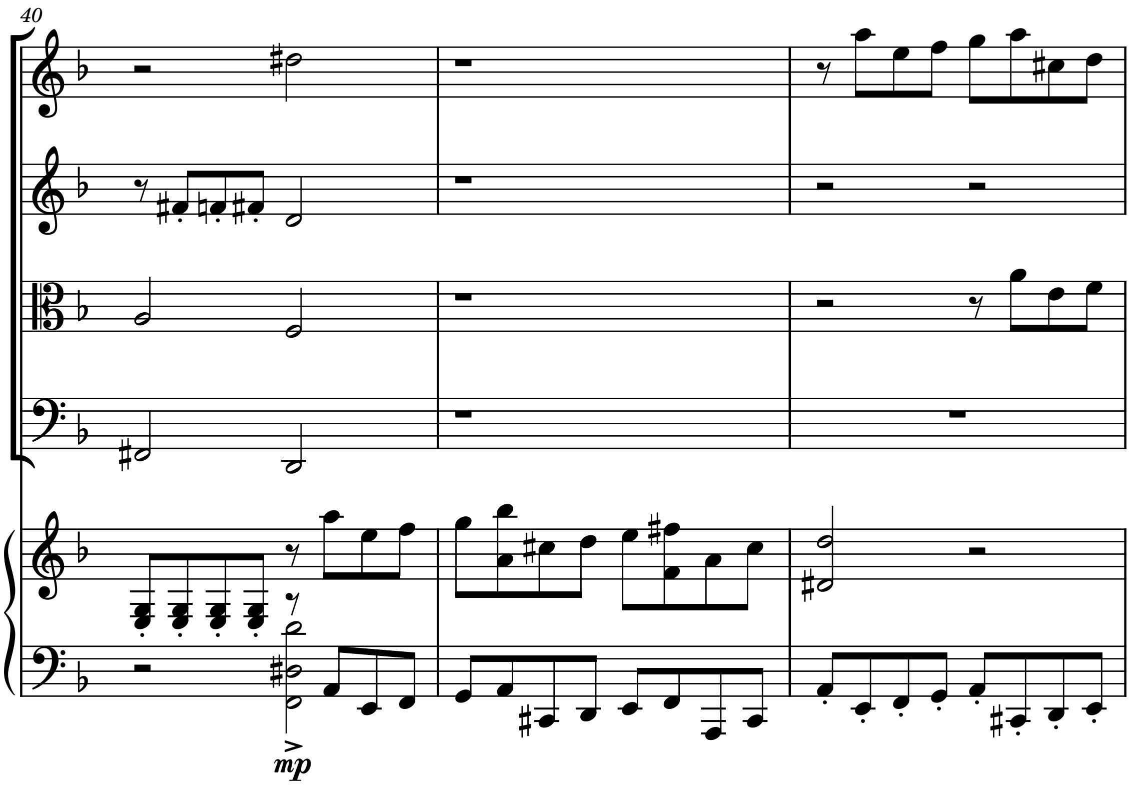

21

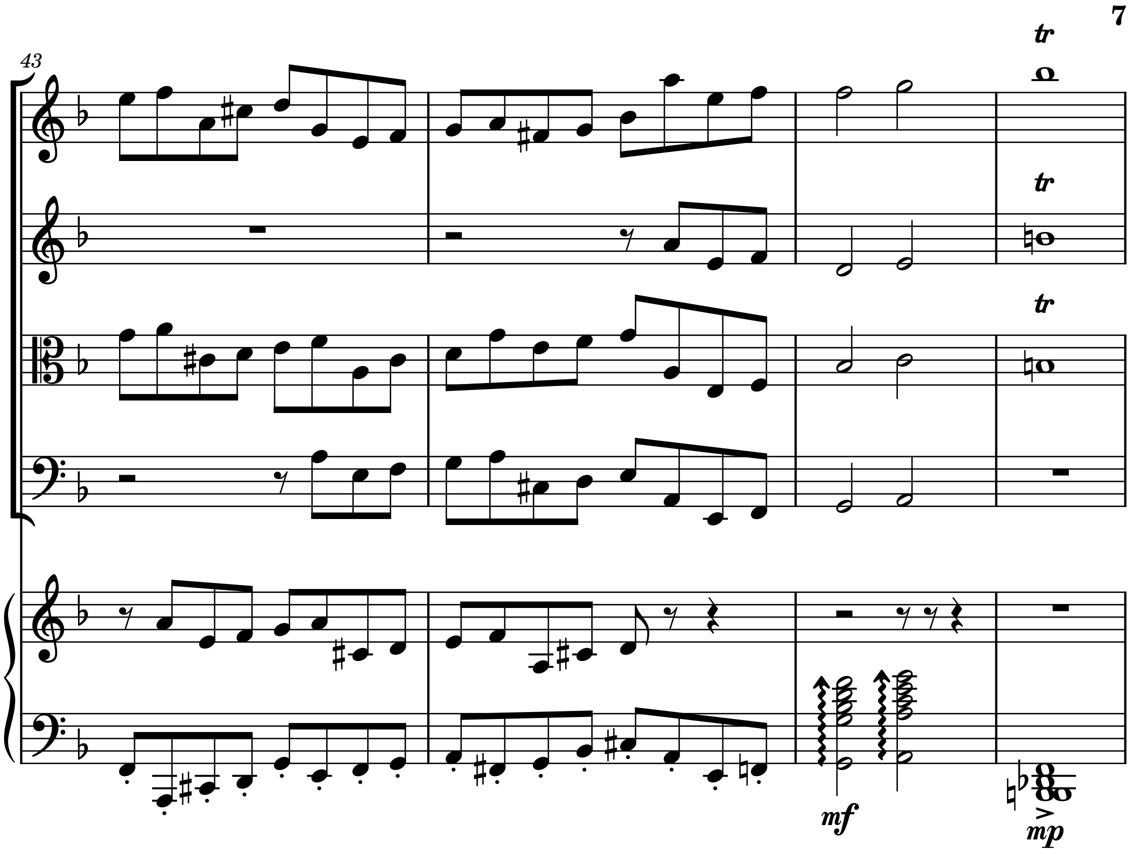

25
mp
mp
mp
mp

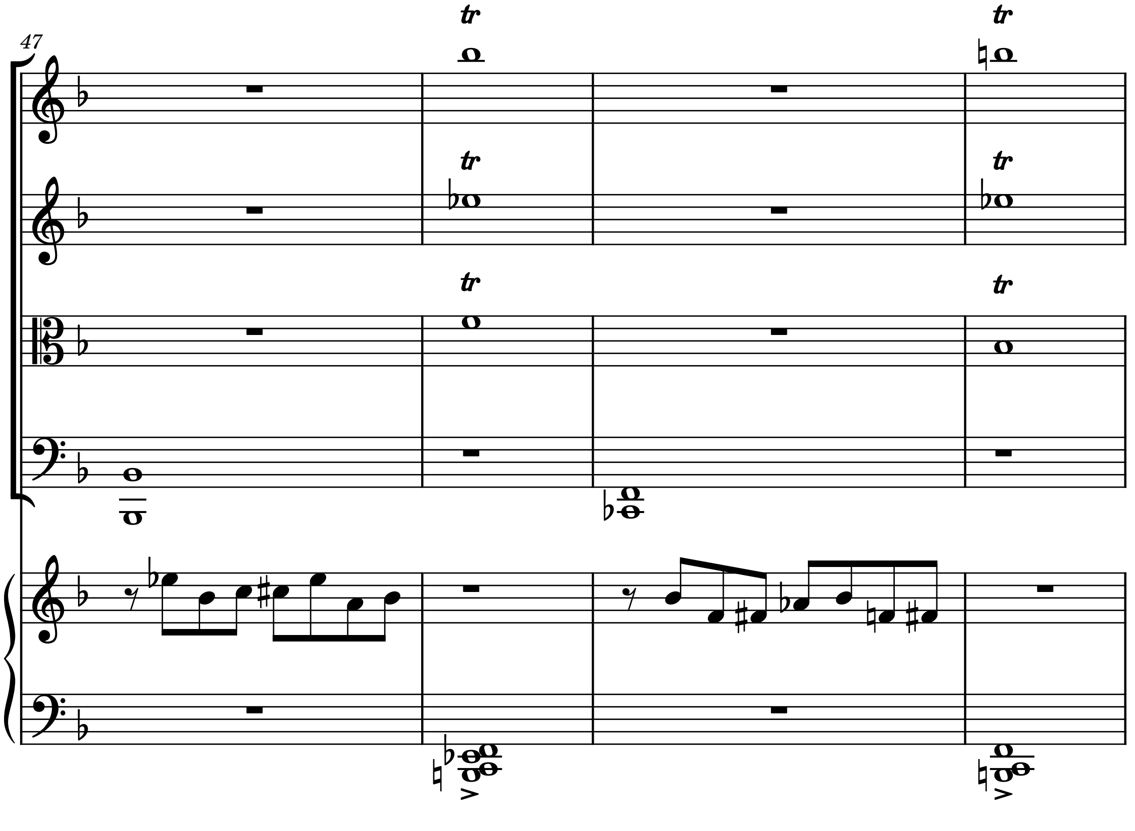

32
To A. Schnittke
mf
mf
mf
mf

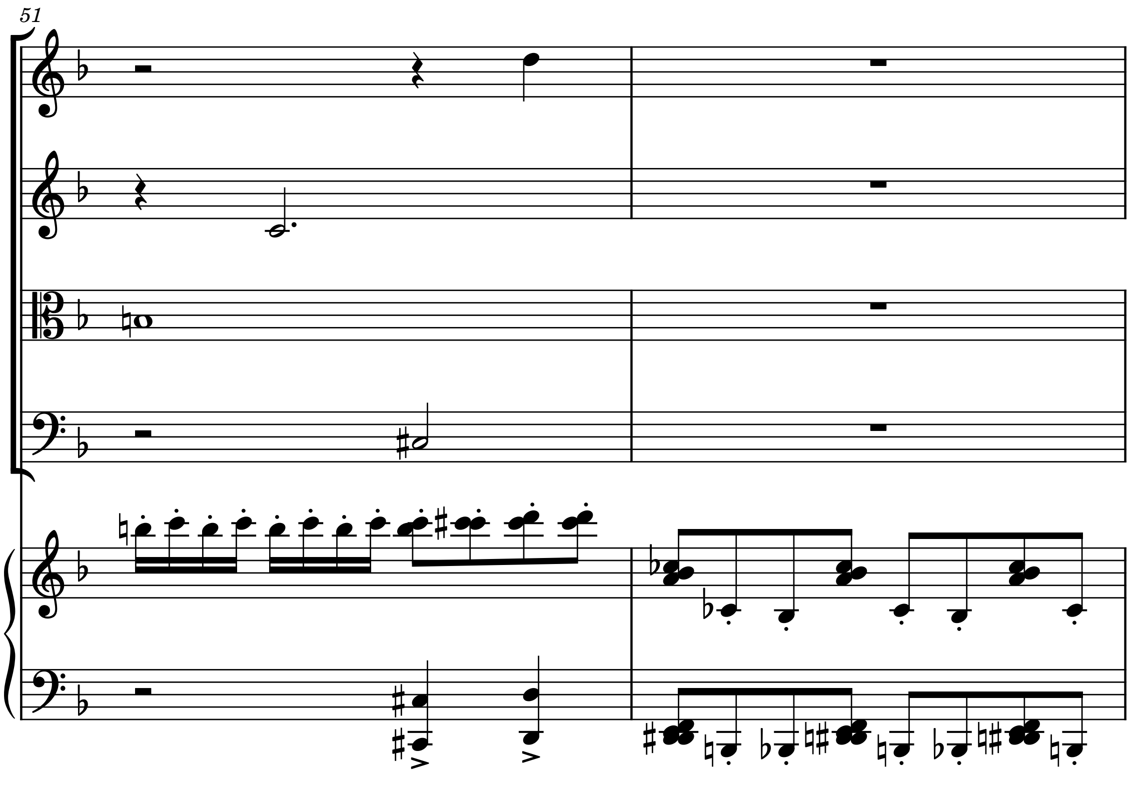

35

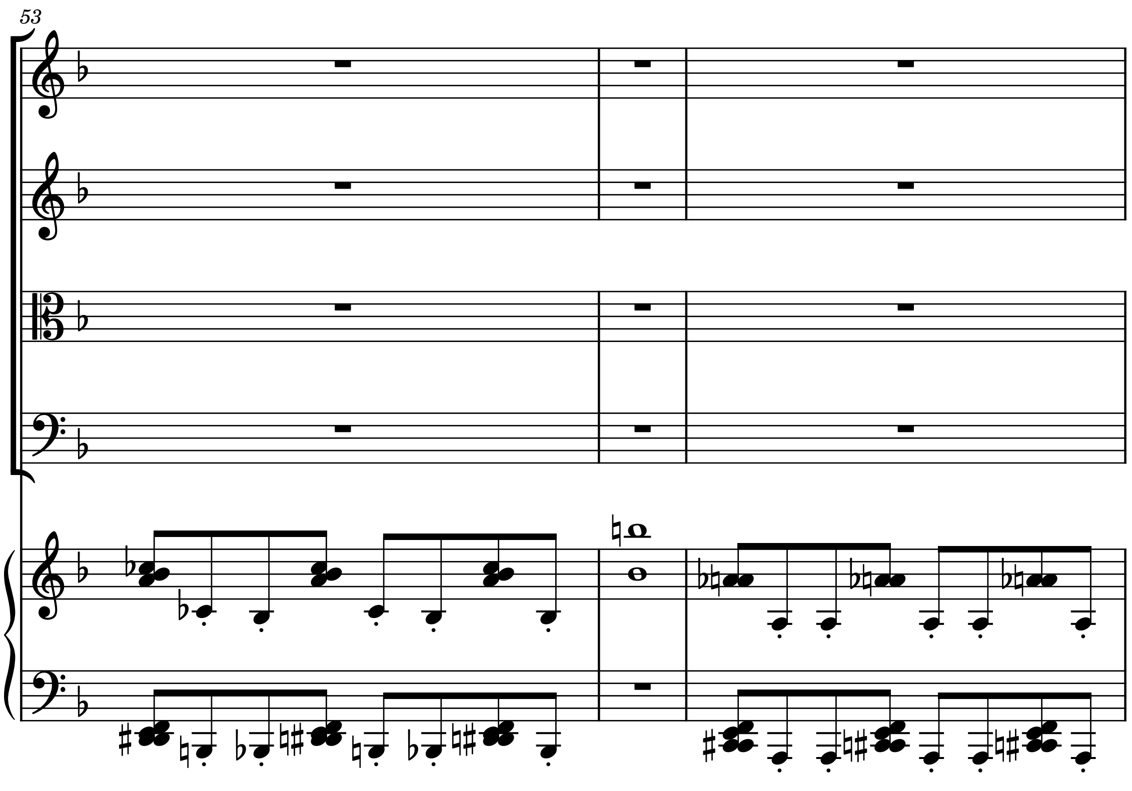

38
p

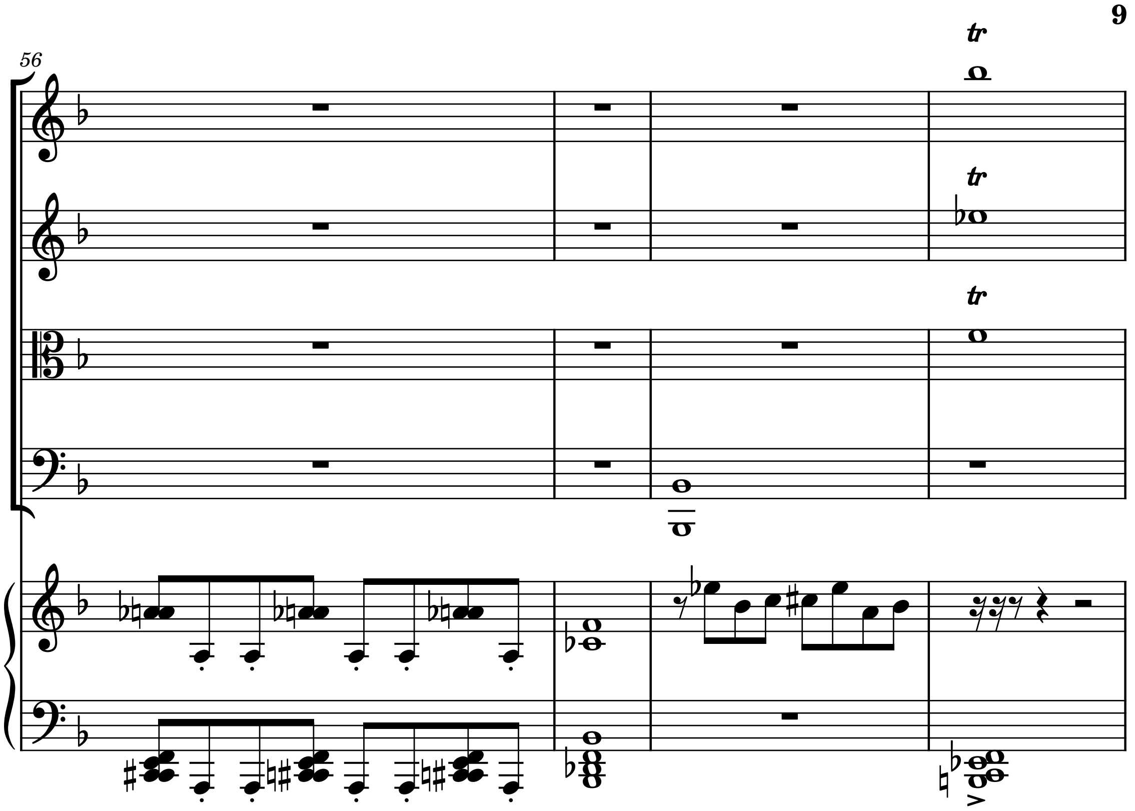

40
mp


43
tr
tr
tr
mf
mp

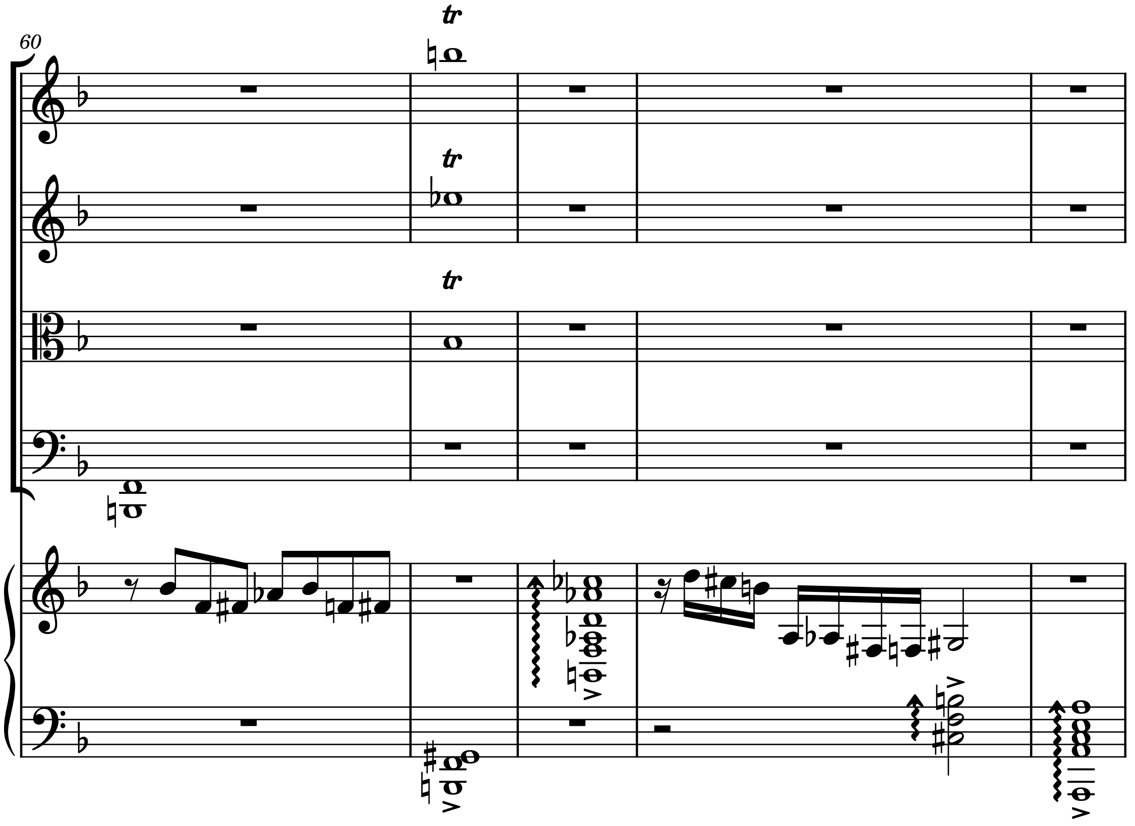

47
tr
tr
tr
tr
tr
tr



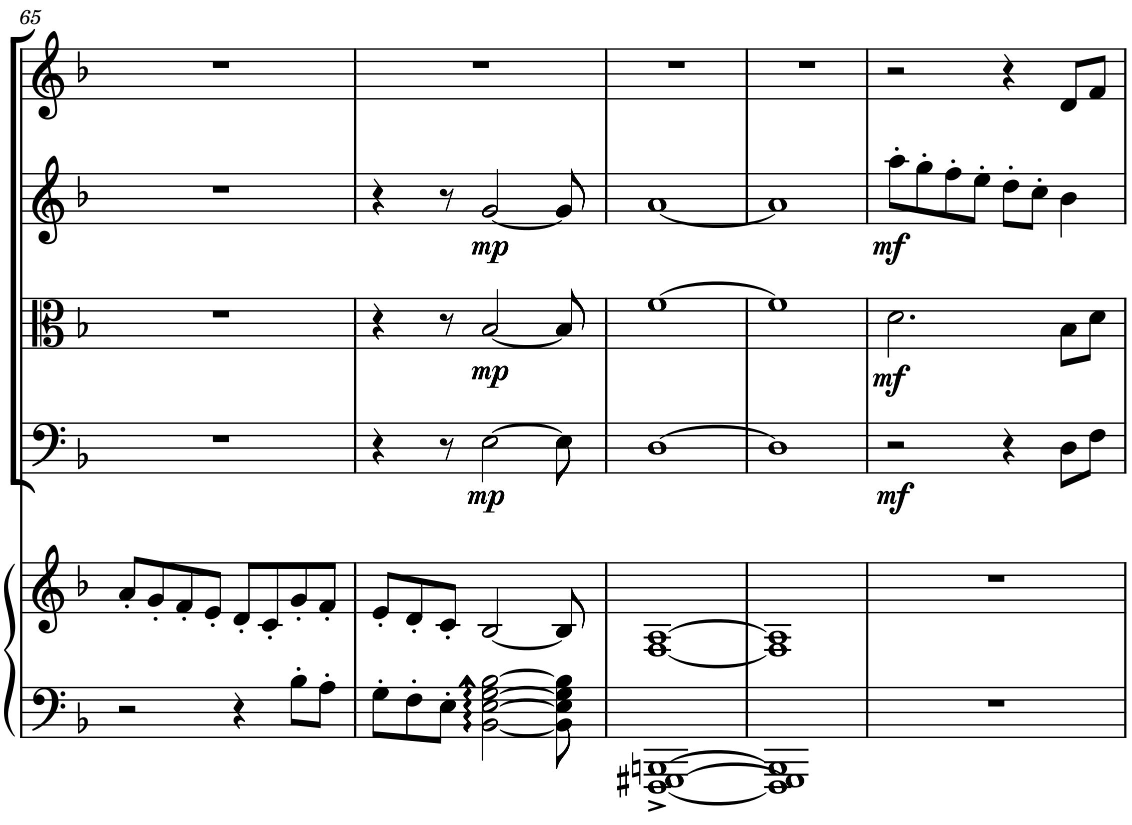


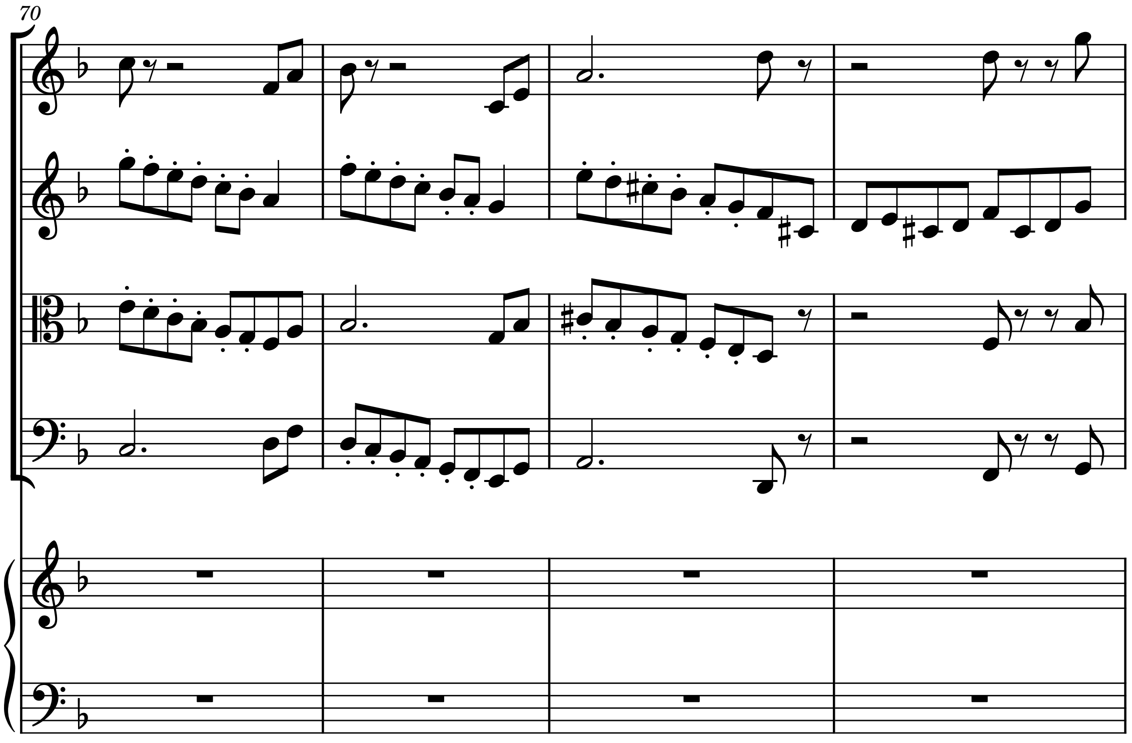


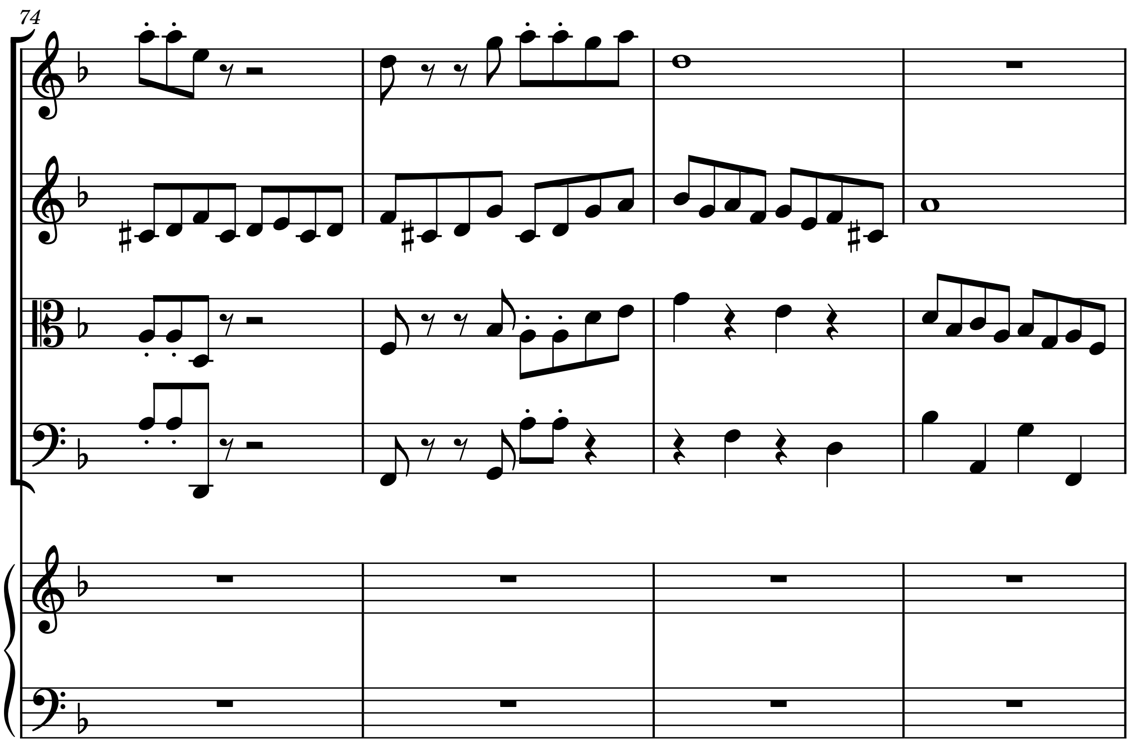
56
tr
tr
tr

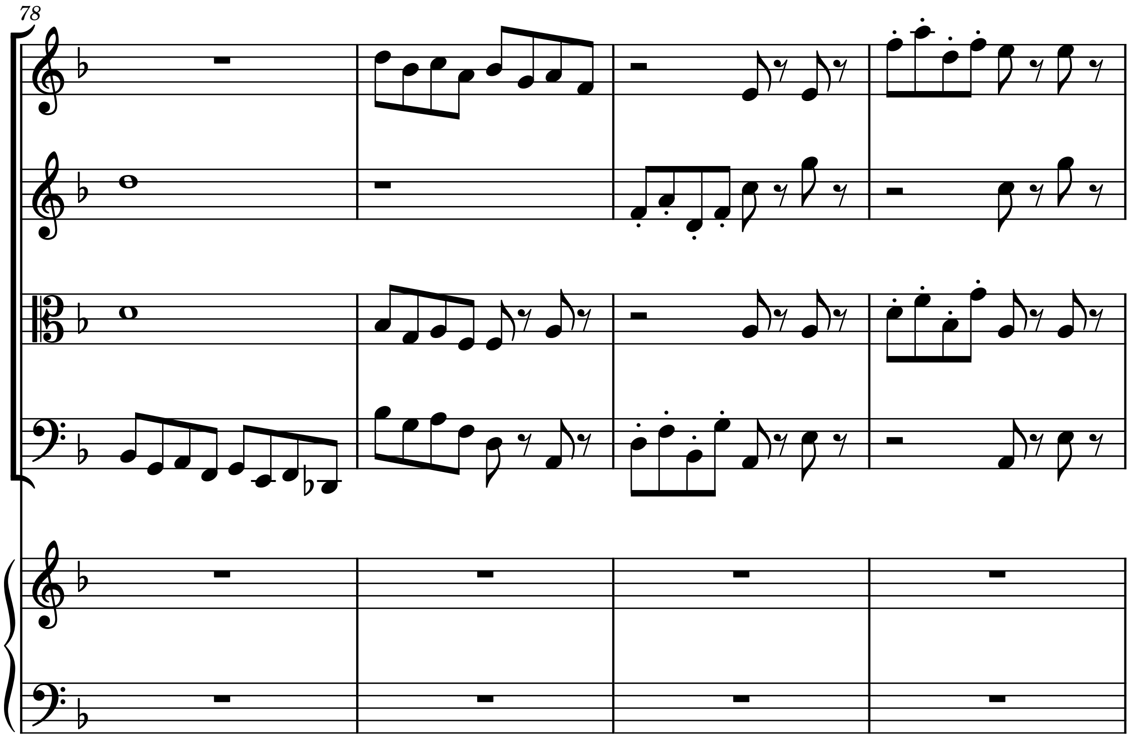


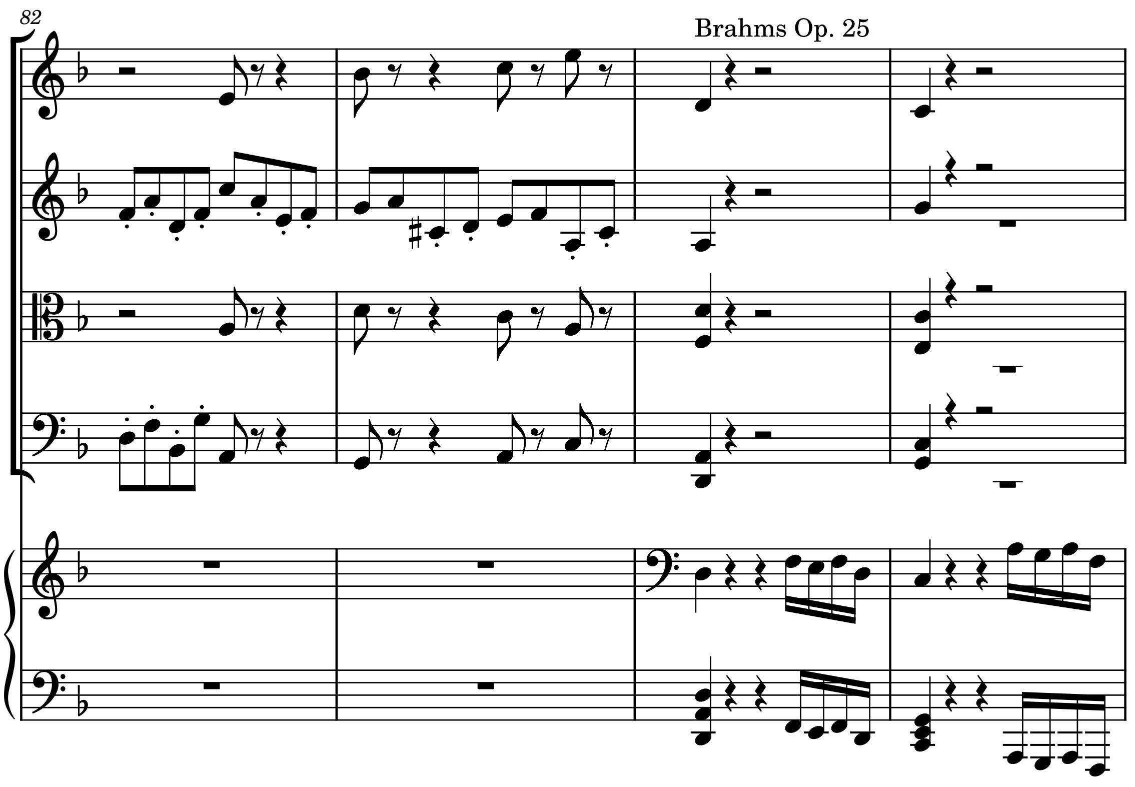
60
tr
tr
tr

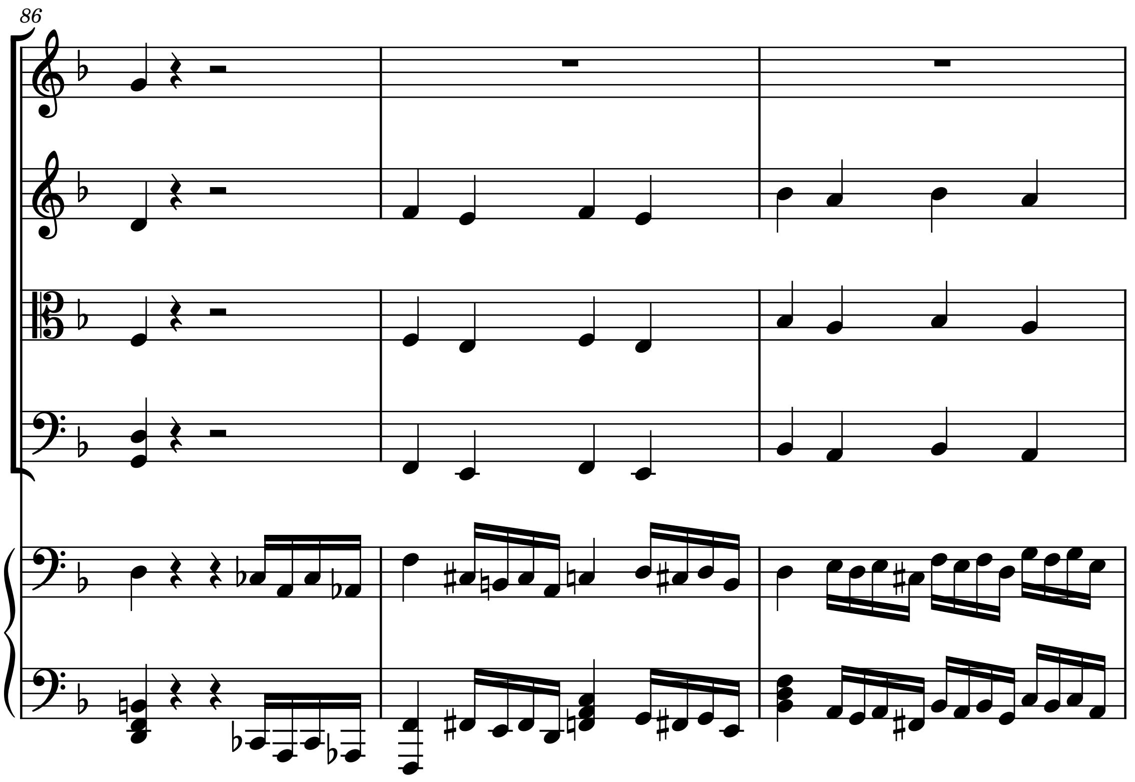

65
mp
mp
mp
mf
mf
mf

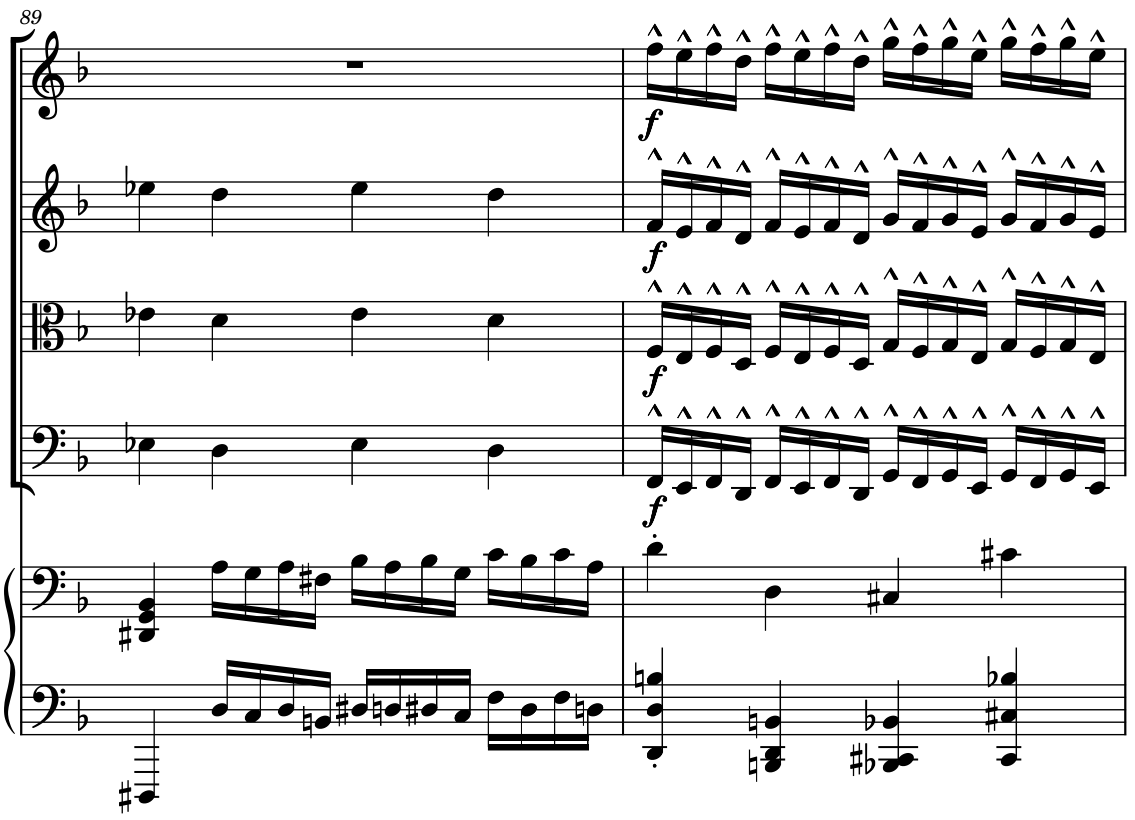

70

74


78



82
Brahms Op. 25


86

89


91