Università degli Studi di Cagliari
Dept. of Mathematics and Informatics
PhD School in Informatics

Doctoral Thesis

A Semantic Deconstruction of Session Types

Supervisor:
Massimo Bartoletti

Author:
Alceste Scalas

PhD school coordinator:
G. Michele Pinna

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy

May 2015
Academic Year 2013–2014

Declaration of Authorship

I, Alceste Scalas, declare that this thesis titled *"A Semantic Deconstruction of Session Types"* and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at the University of Cagliari.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at the University of Cagliari or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Date: May 4th, 2015

"It is not the task of the University to offer what society asks for, but to give what society needs."

Edsger W. Dijkstra

Università degli Studi di Cagliari
Dept. of Mathematics and Informatics
PhD School in Informatics

Abstract

Doctoral Thesis

A Semantic Deconstruction of Session Types

by Alceste Scalas

This work investigates the semantic foundations of *binary session types*, by revisiting them in the abstract setting of labelled transition systems. The main insights and contributions are:

- a semantically unified approach to the study of session types and CCS processes with *synchronous* and *asynchronous* semantics — the latter obtained with the addition of unbounded *buffers*;
- a semantic approach to *safety*, based on a syntax-independent characterisation of *deadlock states*, *orphan messages* and *unspecified reception* configurations;
- an *I/O compliance relation* between generic behaviours, that we demostrate to be sound and complete w.r.t. safety in asynchronous session types;
- an *I/O simulation relation* between generic behaviours, which generalises the usual syntax-directed notions of typing and subtyping, encompassing synchronous and asynchronous session types;
- a proof-of-concept syntax-driven type system developed from the semantic setting through a (partial) axiomatisation of I/O simulation.

This work extends the session types theory to some common programming patterns which are not typically addressed in the session types literature, and aims at setting the ground for further improvements.

Acknowledgements

First and foremost, I wish to thank my PhD supervisor, Massimo Bartoletti, for his invaluable advice, for the long discussions in front of the whiteboard, and for all the opportunities he gave me to learn new things. The last 3 years have been intense and stimulating, and they certainly represent a pivoting point of my life — both professionally and personally. Massimo's role and influence is difficult to overstate.

I also wish to thank Roberto Zunino: our looong Skype meetings have been a fundamental part of my PhD activity, and they have had a huge impact on my work.

I am grateful to Simon Gay and Luca Padovani, for reviewing this thesis and giving detailed remarks and helpful suggestions.

Thanks to Emilio Tuosto and Nobuko Yoshida for having me as a visiting student at their respective institutions, and for the insightful discussions and fruitful collaboration. During such visits, I also had the opportunity to work with Julien Lange: I am glad that this happened not just once, but twice!

Thanks to my (ex-)colleagues at the University of Cagliari — in particular, Giovanni Casu, Tiziana Cimoli, Paolo Di Giamberardino, Maurizio Murgia. Thanks to G. Michele Pinna, for his work as coordinator of the PhD school in Informatics, and for his availability and helpfulness.

Thanks to all the members of the Mobility Reading Group at Imperial College London: I learned a lot just by seeing the way they work.

Thanks to my mother and father, who helped and supported me throughout this endeavour. Thanks to all my friends, who encouraged me to take the PhD challenge, and bore with me when I suddenly disappeared from social life (especially under deadlines).

Last but not least, I wish to say thanks to Ivana and Emiliano, and to Giacomo, Letizia, Edoardo and Emanuele: you have been a model and an inspiration.

This thesis is the outcome of a 3-year PhD grant financed by the Italian Ministry of University and Research (MIUR) — ex D.M. n. 198/2003, "ICT e componentistica elettronica".
This work has also been partially supported by: Aut. Reg. of Sardinia L.R.7/2007 CRP-17285 (TRICS); P.I.A. 2010 ("Social Glue"); MIUR PRIN 2010–11 project "Security Horizons"; EU COST Action IC1201 "Behavioural Types for Reliable Large-Scale Software Systems" (BETTY).

Contents

List of Figures

List of Tables

List of Code Samples

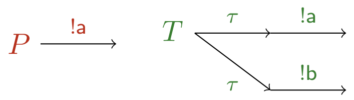
Chapter 1

Introduction and motivation

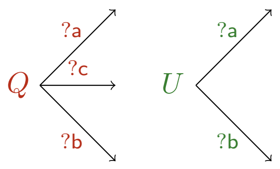
Session typing is a well-established approach to the problem of correctly designing distributed applications [HVK98; HYC08; THK94]. In a nutshell, a (dyadic) session type is the specification of a communication protocol, and describes how a process is expected to use a bi-directional communication channel with another process. In order to achieve correct interaction, each endpoint of the channel is associated with a session type, so that the two endpoint types are *dual* of each other: this, intuitively, means that whenever one endpoint is expected to send some data, the other endpoint will be waiting to receive it — and *vice versa*. A session type specifies which input/output messages are allowed in each protocol state, what type of data is carried by each message, and which protocol state is reached after a message is sent/received: hence, a session type essentially describes a state transition system. The correspondence between a session type and the behaviour of a process (usually specified in a dialect of the π-calculus [MPW92; SW01]) is established via *static type checking* — i.e., by examining the body of the process through a set of syntax-driven rules. If the type checking succeeds, then the send/receive actions performed by the process at runtime will not diverge from those described by the session type: as a consequence, if *both* processes interacting through a channel respect their endpoint's session type, then their interaction will proceed correctly.

Session types are a particular case of *behavioural types* — and the usual technical tool used to prove the correctness of a behavioural type system is *subject reduction*. Say P is a process that communicates through a channel, and T is a session type it should respect. Roughly, subject reduction guarantees that, if we have a typing judgement $\vdash P : T$, then whenever P takes a computation step $P \xrightarrow{\ell} P'$, also the type can take a similar step, i.e. there exists some T' such that $T \xrightarrow{\ell} T'$ and $\vdash P' : T'$.

This relation between processes and types somehow resembles the *simulation* relation in *Labelled Transition Tystems (LTSs)* [Mil89]: a state T simulates a state P iff, whenever $P \xrightarrow{\ell} P'$, then $T \xrightarrow{\ell} T'$, for some T' which still simulates P'. This seems to suggest that the judgement $\vdash P : T$ is rooted in some kind of "process-type simulation". To elaborate further on this insight, consider a session type $T = \mathsf{!a} \oplus \mathsf{!b}$, which models an *internal* choice between two outputs. We can implement this session type with the process $P = \mathsf{!a}$ which just wants to output $\mathsf{!a}$. Intuitively, the process P respects the type T, because any client who can handle both choices in T will interact "correctly" with P. Now, let us consider the transition diagrams of P and T (on the left): we can observe that P is (weakly) simulated by T, in symbols $P \precapprox T$, because each move of P is matched by a move of T.



Let us now consider the type $U = \mathsf{?a} \,\&\, \mathsf{?b}$, which models an *external* choice between two inputs, and let $Q = \mathsf{?a} + \mathsf{?b} + \mathsf{?c}$ (where $+$ is the standard CCS choice operator) which allows for an additional input $\mathsf{?c}$. Again, Q respects U: any client compatible with U will not exploit the additional choice, and will interact "correctly" with Q. But let us look at the LTSs of Q and U (on the right): differently from the previous case, now we have that Q is *not* weakly simulated by U (whereas the converse $U \precapprox Q$ holds). This shows that the weak simulation relation does not faithfully capture the notion of session typing: indeed, the previous examples suggest that a hypothetical "process-type simulation" should treat input and output capabilities differently: intuitively, it should be *covariant* w.r.t. outputs and *contravariant* w.r.t. inputs.



A similar kind of co/contra-variance arises when dealing with *subtyping*. The intuition is that if a session type T is subtype of U, and we have two processes P, Q such that $\vdash P : T$ and $\vdash Q : U$, then P can safely "replace" Q: i.e., each process that interacts "correctly" with Q will also interact "correctly" with P. Again, the session subtyping relations (e.g. [GH99; GH05; CHY07; CHY12]) are *covariant w.r.t. outputs and contravariant w.r.t. inputs* (or *vice versa*, depending on their approach to the problem[1]); moreover, they

[1]The difference between these approaches, which gives rise to opposite but consistent orderings, will be outlined in Section 9.4.

are *coinductive*. This suggests a link between the subtyping relation and our hypothetical "process-type simulation".

1.1 Some questions

The intuition of a behavioural relation underlying typing and subtyping suggests several questions and research topics, that we will articulate in the next sections.

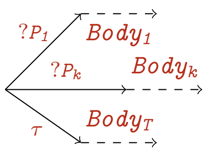
1.1.1 Can we reason on session-based interactions in a language-independent way?

Several papers have studied session typing relations (e.g. [Bet+08; Boc+10; CV10; Hon93; HVK98; HYC08; MV11]) and subtype preorders (e.g. [BL10; BL14; BZ07b; BZ08; Car+06; CP09b; GH99; GH05; BH14; BL15]). Despite the variety of aims and results, all these works share a common assumption: the existence of some specific language for types and/or processes. Such an assumption manifests itself in different ways: typing/subtyping definitions are often purely *syntax-driven* (e.g. in the form of a type system, or a syntax-based coinductive subtyping relation in [GH99; GH05]); otherwise, when *behavioural* definitions are provided for subtyping and/or process-type conformance (e.g. [BZ08; Car+06; CP09b; Pad12; Pad13; BH14; BL15]), it is still assumed that the behaviours under study are generated by some given process/type language. This seems in slight contrast with a common principle in concurrency theory: keeping syntax separated from semantics. Indeed, behavioural equivalences (e.g. (bi)simulation, testing, *etc.*) are typically defined over *arbitrary* LTSs, and then applied to specific calculi by providing the latter with LTS semantics [San12]. By following this last principle, can we find syntax-independent behavioural relations that, once applied to specific process calculi, correspond to the usual typing/subtyping rules outlined above?

1.1.2 Can we devise new syntactic typing rules as instances of some general relation?

A drawback of the syntax-driven approaches to session types is that they do not usually consider some common programming patterns for interactive applications. For example, let us think about a server waiting for client's input: typically, the server must handle the case where such inputs do not arrive. This can be achieved via signals/exceptions handling, or other programming language constructs. In Erlang [Eri15], for instance, one can write:

```
receive  P1 -> Body1...
         Pk -> Bodyk
after    10 -> BodyT
```



This causes `receive` to be aborted if no messages matching the patterns $P_1, \ldots, P_k$ arrive within 10 milliseconds; in this case, $Body_T$ is executed — where the program may e.g. do some internal actions and start receiving again. Such a program blurs the distinction between internal/external choices: intuitively, its LTS (on the right) has a process state with external inputs $?P_1, \ldots, ?P_k$ and an internal τ-move abstracting the timeout. This eludes the notion of *"structured communication-based programming"* at the roots of the session types approach [Hon93; HVK98]; yet, it is a use case that one would like to somehow typecheck to ensure "correct" interaction. This pattern cannot be usually written in standard session calculi, since they do not include a free τ-prefix as part their grammar: this may be due to the desire for a minimal and manageable theory, or to some inherent technical difficulty in decoupling the structure of a process from the structure of the types. Can a more syntax-independent approach handle this case, and possibly others?

1.1.3 Can we reason on synchronous and asynchronous interactions in an uniform semantic setting?

Another limitation of syntax-based approaches arises in the handling of *asynchrony*. Real-world distributed systems usually adopt *FIFO-buffered* interactions: in the TCP/IP protocol, for instance, output messages are sent by each process in a *non-blocking* fashion, and are delivered to the recipient after traversing intermediate message queues. Now, consider a type $T = !\mathsf{a}.?\mathsf{b}$, where $?\mathsf{b}$ is fired *after* $!\mathsf{a}$: the "natural" corresponding process is $P = !\mathsf{a}\,.\,?\mathsf{b}$, which performs the expected actions in the same order. However, in an *asynchronous* setting, the sequentiality of the resulting interactions cannot be enforced: i.e., if P fires $!\mathsf{a}$ asynchronously, then it will immediately start waiting for $?\mathsf{b}$, without knowing whether $!\mathsf{a}$ is still buffered, or was actually received by the other endpoint. Hence, one might argue that, in the presence of buffers, also $P' = ?\mathsf{b} \mid !\mathsf{a}$ (where $?\mathsf{b}$ and $!\mathsf{a}$ are executed in parallel) "asynchronously conforms" to T: the intuition would be that, in the presence of asynchrony, T does *not* actually require that a synchronisation on ?b happens strictly after a synchronisation on !a. Several works on session types study asynchronous semantics (e.g. [NT04; GV07; CDY07; BC08; HYC08; MYH09; Mos09; CDY14]) and the possibility of relating P with T is standard. However, to the best of our knowledge, the relation between P' and T is not addressed. In Section 1.1.3, we wondered whether "missing cases" like this one might be due to a desire for minimality, or technical difficulties related to the structural difference between P' and T. But in order to reason on this issue, we need to address a more fundamental question: how can we formalise the intuition that *both* P and P' *"correctly asynchronously implement"* T, and thus show that (in the presence of asynchrony) T is an "asynchronous type" for *both* P and P'? Moreover, can we address this problem in the *same* semantic framework outlined in Sections 1.1.1 and 1.1.2?

1.1.4 Is there a language-independent notion of "correct" interaction?

In the discussion so far, we relied on the intuition of a "correct" interaction between processes. This "correctness" can be defined in different ways, depending on the calculus in use, and what is considered to be an "error". A typical notion of "correct" interaction is *progress*, which intuitively holds when two processes keep interacting until they *both* terminate — i.e., we have an error when the interaction stops because one process exposes transitions on which the other cannot synchronise. This notion is widespread in concurrency theory, and can also be found in session types literature (e.g. [BL10; BL14]). Different (but related) notions of error are considered in the setting of Communicating Finite State Machines (CFSM) [BZ83]. There, the explicit handling of asynchrony (via FIFO buffers) allows to characterise error conditions which are finer than lack of progress, such as *orphan messages* [LTY15; DY13] (i.e., messages that are sent and buffered, but never received — also studied for session types in [CDY14]) and *unspecified reception configurations* [CF05] (which arise when a machine is waiting for some input, but no matching message is ever sent). Assuming that the treatment of asynchrony outlined in Section 1.1.3 is achieved, is it possible to *semantically* characterise a notion of "correct" interaction (and thus, corresponding notions of erroneous communication) which is language-independent and fits in the same behavioural framework of our (hypothetical) typing/subtyping relation?

1.1.5 Towards a general approach

Some of the topics in Sections 1.1.1 to 1.1.4 (in particular, those related to process/type syntax) might be addressed by extending some previous work (e.g., by adding syntactic constructs and *ad hoc* typing rules to some existing process/type language, and proving subject reduction). In this work, however, we tackle them under a more general approach: we *revisit the semantic foundations of session types*, in a behavioural framework allowing the treatment of *asynchrony*; we aim for language-independent relations and properties that can be later applied to specific process calculi and programming languages.

1.2 Contributions and structure of the work

In this work, we study a behavioural theory of session types, aimed at unifying the notions of typing and subtyping, including both synchronous and asynchronous semantics.

We represent session-based interactions in a simple but general way: we study *generic first-order LTS processes* (called *behaviours*) whose transition labels are either inputs, outputs, or silent (τ). Intuitively, we look at the input/output capabilities that two interacting processes expose *inside* a session, and abstract everything else (e.g. internal moves or communications on other sessions) as τ-moves. Moreover, in order to better focus on the underlying behavioural theory, we abstract from transmitted/received values by letting labels represent *data sorts* — and in this way, we also neglect the distinction between *choice labels* and *carried types* usually found in session types syntax[2]. To the best of our knowledge, this is the first work that approaches the session types theory (taking several ideas from the related fields of *behavioural contracts* and *Communicating Finite State Machines*) by studying arbitrary behaviours within an LTS, not necessarily generated by some specific process language.

A crucial technical choice is the *asymmetric treatment of input and output labels*, that, albeit dual of each other, are generally *not* interchangeable in our main developments. This choice is inspired by the semantics of CFSMs, and is contrary e.g. to the full symmetry between actions and co-actions typically found in CCS-based literature.

Moreover, again to the best of our knowledge, this work is unique in its *treatment of synchronous and asynchronous behaviours*: we handle them within the same unified

[2]These abstractions will be further discussed later, in Sections 9.3 and 9.8.

semantic framework, and study and compare them using the same relations. With this approach, we are able to keep a remarkably simple notation (e.g., we do not require dedicated labels to signal message buffering and consumption), and obtain several results about the *preservation of relations when passing from synchronous to asynchronous semantics*, where unbounded buffers can turn finite-state processes into infinite-state ones.

The following list describes the structure of the work and the main technical contributions:

- we start in Chapter 2 by setting our semantic, syntax-independent framework (*Input/Output Labelled Transition Systems*) and by introducing two process algebras that we will use throughout our treatment: session types (Definition 2.14) and CCS (Definition 2.23). We equip both of them with a synchronous and an asynchronous semantics — the latter obtained by pairing terms with *unbounded FIFO buffers*;
- in Chapter 3 we give a running example, that we will reuse throughout our treatment;
- in Chapter 4 we define the *I/O compliance* relation $\ddot{\bowtie}$ as a notion of "correct" interaction between behaviours, stricter than progress, albeit coinciding with it on synchronous session types (Theorem 4.9). We also show that $\ddot{\bowtie}$ is preserved when passing from synchronous to asynchronous session behaviours (Proposition 4.12);
- in Chapter 5, we generalise to our semantic setting the classical notion of *safety*, intended as the lack of *deadlock state* (Definition 5.1), *orphan message* (Definition 5.5) and *unspecified reception configuration* (Definition 5.15). We can then formally justify the "correctness" of I/O compliance claimed in Chapter 4, by proving that $\ddot{\bowtie}$ is sound w.r.t. safety (Theorem 5.22), and also complete in the case of asynchronous session behaviours (Theorem 5.24);
- in Chapter 6 we introduce the *I/O simulation* relation $\ddot{\leqslant}$ between behaviours. We show that it is an I/O compliance-preserving preorder (Theorems 6.12 and 6.13), is a subtyping relation [CHY12; GH05] (Theorem 6.16), and is preserved when passing from synchronous to asynchronous session types semantics (Theorem 6.27);
- in Chapter 7 we show that $\ddot{\leqslant}$ can give the semantic basis to induce a syntax-driven type systems based on session types, which guarantee safe interactions without the need of a separate proof of subject reduction (Theorem 7.16);

- in Chapter 8 we present *The LTS Workbench (`LTSwb`)*, an extensible Scala [Oa04] toolbox allowing to define LTSs and processes, and compute relations between their states. `LTSwb` allows to experiment with our semantic framework, and includes an implementation of I/O compliance;
- in Chapter 9 we compare our framework and results with other related works and approaches;
- finally, Chapter 10 contains a summarised view of our work, and proposes some directions for further work.

To relieve the reader from execessively onerous technicalities, we have opted for moving some auxiliary results and some proofs to the appendix.

1.3 Improvements w.r.t. [BSZ14]

This work is an improved and extended version of [BSZ14]. We corrected a mistake appearing in that paper, i.e. the fact that the counterpart of Theorem 6.27 lacked the $\preceq_?$ relation in its hypotheses. Moreover:

- Chapter 5 and the results on safety are completely new;
- the I/O simulation $\ddot{\leqslant}$ is now larger, and thus able to relate more compliance-preserving behaviours;
- the typing relation in Chapter 7 has been also enlarged, and its definition streamlined and simplified, bringing it closer to an algorithm (as discussed in Conjecture 10.1).

Chapter 2

Behaviours

In this chapter we exploit the semantic model of labelled transition systems (LTSs) to provide a unifying ground for the notions developed in the later chapters. We consider I/O LTSs where labels are partitioned into internal, input, and output actions (Section 2.1), and we call *behaviours* the states of such LTSs. Then, we exploit this model to embed four calculi for concurrency: binary session types with synchronous or asynchronous semantics (Section 2.2), and synchronous/asynchronous CCS (Section 2.3).

This chapter lays the ground for tackling the issues discussed in Sections 1.1.1 to 1.1.4. The asynchronous semantics will give the first hints on a peculiar feature of our framework, that will emerge more strongly in the next chapters: inputs and outputs are treated differently (e.g., because only the latter can be buffered for asynchrony), and they are not always interchangeable duals (contrary e.g. to the the standard CCS actions and co-actions). Even though in the rest of the work we will sometimes use session types and CCS to write examples (e.g., in Chapter 3) and to discuss related work, most of the technical notions and results do apply to the general class of behaviours.

2.1 Basics

2.1.1 The I/O LTS

Our treatment is developed within the I/O LTS $\left(\mathbb{U}, \mathsf{A}_\tau, \left\{ \xrightarrow{\ell_\tau} \,\middle|\, \ell_\tau \in \mathsf{A}_\tau \right\}\right)$, where:

- $\mathbb{U}$ is the set of all *behaviours*;

$?\mathsf{a}, ?\mathsf{b}, \ldots \in \mathsf{A}^?$	Input actions	$\mathbb{U}$	Set of all behaviours:
$!\mathsf{a}, !\mathsf{b}, \ldots \in \mathsf{A}^!$	Output actions	$\mathbb{U}_{\mathrm{ST}}$	Sync session behaviours
$\mathsf{A} = \mathsf{A}^? \cup \mathsf{A}^!$	Set of I/O actions	$\mathbb{U}_{\mathrm{aST}}$	Async session behaviours
$\mathrm{co}(\cdot)$	Involution of I/O actions	$\mathbb{U}_{\mathrm{aCCS}}$	Async CCS behaviours
		$p, q, r, \ldots \in \mathbb{U}$	Behaviours
$\ell, \ell', \ldots \in \mathsf{A}$	Actions (visible labels)	$T, U, V \ldots \in \mathbb{U}_{\mathrm{ST}}$	Session types (sync)
τ	Internal action	$T[\sigma], U[\rho], \ldots \in \mathbb{U}_{\mathrm{aST}}$	Session types (async)
$\mathsf{A}_\tau = \mathsf{A} \cup \{\tau\}$	Set of labels	$P[\sigma], Q[\rho], \ldots \in \mathbb{U}_{\mathrm{aCCS}}$	Async CCS behaviours
$\ell_\tau, \ell'_\tau, \ldots \in \mathsf{A}_\tau$	Labels	$\mathbb{P}, \mathbb{Q}, \mathbb{R}, \ldots \subseteq \mathbb{U}$	Sets of behaviours

Table 2.1: Summary of notation.

- A_τ is the set of all *labels*, partitioned into *input actions* $\mathsf{A}^? = \{?\mathsf{a}, ?\mathsf{b}, \ldots\}$, *output actions* $\mathsf{A}^! = \{!\mathsf{a}, !\mathsf{b}, \ldots\}$, and the *internal action* τ;
- $\xrightarrow{\ell_\tau} \subseteq \mathbb{U} \times \mathbb{U}$ is the *transition relation.*

We postulate an involution $\mathrm{co}(\cdot)$ such that $\mathrm{co}(?\mathsf{a}) = !\mathsf{a}$ and $\mathrm{co}(!\mathsf{a}) = ?\mathsf{a}$.

Notation 2.1 (Relations)**.** *We write:*

- $\mathcal{R}^*$ *for the reflexive and transitive closure of a relation* $\mathcal{R}$*;*
- $p\xrightarrow{\ell_\tau}$ *when* $\exists p' \,.\, p \xrightarrow{\ell_\tau} p'$*;*
- $p\rightarrow$ *when* $\exists \ell_\tau \,.\, p\xrightarrow{\ell_\tau}$*;*
- $p \xrightarrow{!} q$ *iff* $\exists \mathsf{a} \,.\, p \xrightarrow{!\mathsf{a}} p'$.

Notation 2.2. *For a set* $L \subseteq \mathsf{A}$*, we define* $L^? = L \cap \mathsf{A}^?$ *and* $L^! = L \cap \mathsf{A}^!$.

Definition 2.3 (Nil behaviour)**.** $\mathbf{0}$ *is a behaviour without transitions, i.e.* $\mathbf{0}\not\rightarrow$.

Definition 2.4 (Parallel composition of behaviours)**.** *For all* $p, q \in \mathbb{U}$*, we define the* parallel composition $p \parallel q$ *as the behaviour in* $\mathbb{U}$ *whose transitions are given by the rules:*

$$\frac{p \xrightarrow{\ell_\tau} p'}{p \parallel q \xrightarrow{\ell_\tau} p' \parallel q} \qquad \frac{q \xrightarrow{\ell_\tau} q'}{p \parallel q \xrightarrow{\ell_\tau} p \parallel q'} \qquad \frac{p \xrightarrow{\ell} p' \quad q \xrightarrow{\mathrm{co}(\ell)} q'}{p \parallel q \xrightarrow{\tau} p' \parallel q'}$$

Table 2.1 summarises the syntactic categories and some notation.

$\xrightarrow{\ell_\tau}$, $\rightarrow$, $\xrightarrow{!}$	Labelled/unlabelled/semi-labelled transition relation (Notation 2.1)
$\Rightarrow \; = \; (\xrightarrow{\tau})^*$	Weak τ transition (Definition 2.5)
$\overset{\ell_\tau}{\Longrightarrow} = \Rightarrow\xrightarrow{\ell_\tau}\Rightarrow$	Weak transition relation (Definition 2.5)
$\overset{??\mathsf{a}}{\Longrightarrow}$, $\overset{!!\mathsf{a}}{\Longrightarrow}$	Weakly persistent inputs/outputs (Definition 2.6)
$\Rrightarrow$	Set transition relation (Definition 2.7)
$p\Downarrow = \left\{\ell \,\middle|\, p\overset{\ell}{\Rightarrow}\right\}$	Weak barbs (Definition 2.8)
$p\Downarrow^{?}$, $p\Downarrow^{!}$	Weak input/output barbs (Definition 2.8 and notation 2.2)
$p\Downarrow^{??}$, $p\Downarrow^{!!}$	Weakly persistent input/output barbs (Definition 2.9)
$\cong$	Isomorphism (Definition 2.10)
$\sim$	Strong bisimilarity (Definition 2.11)
$\approx$	Weak bisimilarity (Definition 2.12)
$\precapprox$	Weak similarity (Definition 2.12)

Table 2.2: Summary of transition relations, barbs, and observational relations.

2.1.2 Transition relations

We introduce below the main relations which will be used along the treatment. Some of them, like e.g. weak inputs and outputs, are standard, while some others, like e.g. weakly persistent inputs and outputs, are not, and they will be used later on in our technical development. Table 2.2 summarises these relations, providing pointers to where they are introduced.

Definition 2.5 (Weak transitions)**.** *We define the relation $\Rightarrow$ as the reflexive and transitive closure of $\xrightarrow{\tau}$. We define $\overset{\ell_\tau}{\Longrightarrow}$ as $\Rightarrow\xrightarrow{\ell_\tau}\Rightarrow$. By extension, given a sequence of actions $w = \ell_1, \ldots, \ell_n$, we write $\overset{w}{\Rightarrow}$ for $\overset{\ell_1}{\Rightarrow} \cdots \overset{\ell_n}{\Rightarrow}$. Further, we write $\overset{!}{\Rightarrow}$ for $\Rightarrow\xrightarrow{!}\Rightarrow$, we write $\mathbb{Q} \overset{\ell_\tau}{\Rightarrow} q$ whenever $\exists p \in \mathbb{Q} \;.\; p \overset{\ell_\tau}{\Rightarrow} q$, and similarly for $\mathbb{Q} \Rightarrow q$.*

Definition 2.6 (Weakly persistent inputs/outputs)**.** *We write $p\overset{??\mathsf{a}}{\Longrightarrow}$ (resp. $p\overset{!!\mathsf{a}}{\Longrightarrow}$) iff for all p', $p \Rightarrow p'$ implies $p'\overset{?\mathsf{a}}{\Longrightarrow}$ (resp. $p'\overset{!\mathsf{a}}{\Longrightarrow}$).*

The *set transition relation* $\Rrightarrow$ will be used later on in Chapter 6 to define I/O simulation.

Definition 2.7 (Set transition relation)**.** *We write $q \Rrightarrow \mathbb{Q}$ iff $\emptyset \neq \mathbb{Q} \subseteq \{q' \mid q \Rightarrow q'\}$. We write $\mathbb{Q} \Rrightarrow \mathbb{Q}'$ iff $\forall q' \in \mathbb{Q}' \,.\, \exists q \in \mathbb{Q} \,.\, q \Rightarrow q'$.*

2.1.3 Barbs

Definition 2.8 (Weak barbs)**.** *We define the* weak barbs of p *as:*

$$p\Downarrow \;=\; \left\{\ell \,\middle|\, p\overset{\ell}{\Rightarrow}\right\}$$

By extension, we write $\mathbb{Q}\Downarrow$ *for* $\bigcup_{q \in \mathbb{Q}} q\Downarrow$.

Definition 2.9 (Weakly persistent barbs)**.** *We define the sets:*

$$p\Downarrow^{??} = \left\{ ?\mathsf{a} \,\middle|\, p \overset{??\mathsf{a}}{\Longrightarrow} \right\} \qquad p\Downarrow^{!!} = \left\{ !\mathsf{a} \,\middle|\, p \overset{!!\mathsf{a}}{\Longrightarrow} \right\}$$

By extension, we write $\mathbb{Q}\Downarrow^{??}$ *for* $\bigcap_{q \in \mathbb{Q}} q\Downarrow^{??}$, *and similarly for* $\mathbb{Q}\Downarrow^{!!}$.

2.1.4 Observational relations

We recall below some standard observational relations between behaviours [Mil89].

Definition 2.10 (Isomorphism)**.** *We write* $p \cong q$ *iff the transition diagrams of* p *and* q *are* isomorphic, *i.e. equal up-to node renaming.*

Definition 2.11 (Strong bisimulation)**.** *We say that* $\mathcal{R}$ is a (strong) bisimulation relation *iff, whenever* $p \,\mathcal{R}\, q$*:*

(i) $p \xrightarrow{\ell_\tau} p'$ *implies* $\exists q' \,.\, q \xrightarrow{\ell_\tau} q' \wedge p' \,\mathcal{R}\, q'$;

(ii) $q \xrightarrow{\ell_\tau} q'$ *implies* $\exists p' \,.\, p \xrightarrow{\ell_\tau} p' \wedge p' \,\mathcal{R}\, q'$.

We write $\sim$ *for the largest bisimulation relation. When* $p \sim q$*, we say that* p *and* q *are* bisimilar*.*

Definition 2.12 (Weak simulation)**.** *We say that* $\mathcal{R}$ is a weak simulation relation *iff, whenever* $p \,\mathcal{R}\, q$*:*

(i) $p \xrightarrow{\ell} p'$ *implies* $\exists q' \,.\, q \overset{\ell}{\Rightarrow} q' \wedge p' \,\mathcal{R}\, q'$;

(ii) $p \xrightarrow{\tau} p'$ *implies* $\exists q' \,.\, q \Rightarrow q' \wedge p' \,\mathcal{R}\, q'$.

We write $\precapprox$ *for the largest weak simulation relation, and* $\approx$ *for the largest symmetric weak simulation relation. When* $p \precapprox q$*, we say that* p *is* weakly simulated *by* q*.*

Proposition 2.13 below reports a well-known and immediate result.

Proposition 2.13. $\cong \; \subsetneq \; \sim \; \subsetneq \; \approx \; \subsetneq \; \precapprox$.

2.2 Session types

A session type is an abstraction of the behaviour of a process interacting with its environment. Many different forms of session types have been proposed, with the most notable classes of *binary* (i.e. involving exactly two agents) and *multi-party* session types. We present a simple version of the former, by slightly adapting (and extending to the asynchronous case) those studied in [BL10].

Formally, a session type is a term of a process algebra featuring *internal choice* $\bigoplus_{i \in I} !\mathsf{a}_i.T_i$, *external choice* $\&_{i \in I} ?\mathsf{a}_i.T_i$ and guarded recursion. In an internal choice, the process is declaring that it will choose one of the branches (say, the i-th), fire the output ($!\mathsf{a}_i$), and then proceed with the continuation T_i. In an external choice, instead, the branch is chosen by the context; the process will wait until receiving the input ($?\mathsf{a}_i$), and then continue as T_i. We stipulate that empty choices (either internal or external) represent successful termination.

Definition 2.14 (Session types)**.** Session types *are terms with the syntax:*

$$T \;\; ::= \;\; \&_{i \in I} ?\mathsf{a}_i.T_i \;\;\big|\;\; \bigoplus_{i \in I} !\mathsf{a}_i.T_i \;\;\big|\;\; \mathrm{rec}_X\, T \;\;\big|\;\; X$$

where (i) the set I is finite, (ii) the actions in internal/external choices are pairwise distinct, and (iii) recursion is guarded. We write $\mathbf{0}$ *for the empty (internal/external) choice, and will usually omit its trailing occurrences (e.g., we will write* $!\mathsf{a}.?\mathsf{b}$ *instead of* $!\mathsf{a}.?\mathsf{b}.\mathbf{0}$*). Unless otherwise specified, we assume session types to be* closed.

We present two semantics for session types: one *synchronous* (Definition 2.15) and one *asynchronous* (Definition 2.16). In both semantics, an internal choice first commits to one of the branches $!\mathsf{a}.T$, before enabling $!\mathsf{a}$. An external choice enables all its actions.

Definition 2.15 (Synchronous session behaviours)**.** *We denote with* $\mathbb{U}_{ST}$ *the set of behaviours of the form* T*, with transitions given by the rules:*

$$\frac{k \in I}{\&_{i \in I} ?\mathsf{a}_k.T_i \xrightarrow{?\mathsf{a}_k} T_k}\;(\textsc{TExt}) \qquad \frac{k \in I \quad |I| > 1}{\bigoplus_{i \in I} T_i \xrightarrow{\tau} T_k}\;(\textsc{TInt})$$

$$\frac{}{!\mathsf{a}.T \xrightarrow{!\mathsf{a}} T}\;(\textsc{TOut}) \qquad \frac{T[{}^{\mathrm{rec}_X T}/_X] \xrightarrow{\ell_\tau} T'}{\mathrm{rec}_X\, T \xrightarrow{\ell_\tau} T'}\;(\textsc{TRec})$$

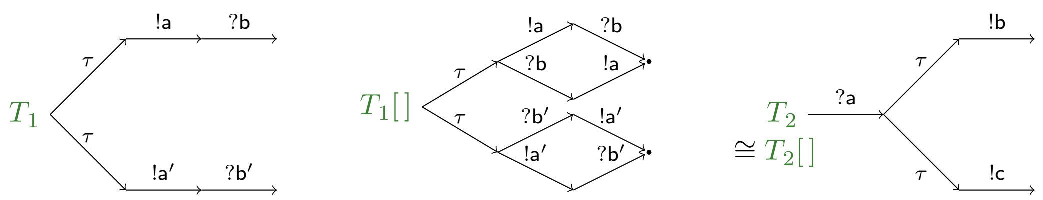


Figure 2.1: Three session behaviours.

For the asynchronous semantics, we consider behaviours of the form $T[\sigma]$ where σ is a sequence of output actions, modelling an unbounded *buffer*. We denote with ϵ the empty buffer, and with $[!\mathsf{a}.\sigma]$ a buffer with head $!\mathsf{a}$ and tail σ.

Definition 2.16 (Asynchronous session behaviours). *We denote with $\mathbb{U}_{aST}$ the set of behaviours of the form $T[\sigma]$, with transition rules:*

$$\frac{k \in I}{\left(\bigoplus_{i\in I} !\mathsf{a}_i.T_i\right)[\sigma] \xrightarrow{\tau} T_k\,[\sigma.!\mathsf{a}_k]}\ (\textsc{TIntA}) \qquad \frac{}{T\,[!\mathsf{a}.\sigma] \xrightarrow{!\mathsf{a}} T\,[\sigma]}\ (\textsc{TOutA})$$

$$\frac{k \in I}{\left(\&_{i\in I} ?\mathsf{a}_i.T_i\right)[\sigma] \xrightarrow{?\mathsf{a}_k} T_k\,[\sigma]}\ (\textsc{CExtA}) \qquad \frac{T\,[\mathrm{rec}_X\,T/X][\sigma] \xrightarrow{\ell_\tau} T'[\sigma']}{\mathrm{rec}_X\,T[\sigma] \xrightarrow{\ell_\tau} T'[\sigma']}\ (\textsc{TRecA})$$

Rule (TIntA) adds the selected output to the end of the buffer, with a τ-move. Rule (TOutA) says that an output $!\mathsf{a}$ at the head of the buffer is consumed with a corresponding $!\mathsf{a}$-transition. The asynchronous rules (TExtA) and (TRecA) are similar to their synchronous counterparts.

Example 2.17. *Let $T_1 = !\mathsf{a}.?\mathsf{b} \oplus !\mathsf{a}'.?\mathsf{b}'$, and $T_2 = ?\mathsf{a}.(!\mathsf{b} \oplus !\mathsf{c})$. Their sync/async behaviours are shown in Figure 2.1. Note that the sync/async behaviours of T_2 are isomorphic.*

For more examples, see Chapter 3.

Similarly to [CDY14], we adopt an *equi-recursive* view of session types [Pie02], through a congruence $\equiv$ which, intuitively, relates session types that are equal up-to unfolding of their recursive subterms (details in Definition A.1 and Proposition A.2). From the semantic viewpoint, equivalent terms are strongly bisimilar (both synchronously and asynchronously), and thus observationally indistinguishable.

Proposition 2.18 below shows that the syntactic form of synchronous session types can be determined by observing their transitions.

Proposition 2.18. *For all T:*

(i) $T \xrightarrow{?\mathsf{a}}$ *iff* $T \equiv ?\mathsf{a}.T' \mathbin{\&} \&_{i \in I} ?\mathsf{b}_i.T_i$;

(ii) $T \xrightarrow{\tau}$ *iff* $T \equiv \bigoplus_{i \in I} !\mathsf{b}_i.T_i$, *with* $|I| > 1$;

(iii) $T \xrightarrow{!\mathsf{a}}$ *iff* $T \equiv !\mathsf{a}.T'$.

Proof. See page 114. □

The main semantic differences between synchronous and asynchronous session types are formalised in Proposition 2.19 below: roughly, asynchrony turns sequences of outputs into sequences of τs, but input transitions and τ-transitions are preserved. For example, we have:

(i) $?\mathsf{a}.\mathbf{0} \xrightarrow{?\mathsf{a}} \mathbf{0}$ becomes $?\mathsf{a}.\mathbf{0}[] \xrightarrow{?\mathsf{a}} \mathbf{0}[]$ (i.e., input transitions are preserved by buffering);

(ii) $!\mathsf{a}.\mathbf{0} \oplus !\mathsf{b}.\mathbf{0} \xrightarrow{\tau} !\mathsf{a}.\mathbf{0}$ becomes $!\mathsf{a}.\mathbf{0} \oplus !\mathsf{b}.\mathbf{0}[] \xrightarrow{\tau} \mathbf{0}[!\mathsf{a}]$, and similarly for the $!\mathsf{b}$-branch (i.e., the τ-transition of a non-deterministic internal choice is preserved in the asynchronous behaviour — where it signals the buffering of the selected prefix);

(iii) $!\mathsf{a}.!\mathsf{b}.\mathbf{0} \xrightarrow{!\mathsf{a}} !\mathsf{b}.\mathbf{0} \xrightarrow{!\mathsf{b}} \mathbf{0}$ becomes $!\mathsf{a}.!\mathsf{b}.\mathbf{0}[] \xrightarrow{\tau} !\mathsf{b}.\mathbf{0}[!\mathsf{a}] \xrightarrow{\tau} \mathbf{0}[!\mathsf{a}.!\mathsf{b}]$ (i.e., deterministic output transitions are "replaced" with τs in the asynchronous behaviour).

Proposition 2.19 also states that, by observing the transitions and the buffer of an asynchronous session behaviour, we can recover information about the corresponding synchronous behaviour: e.g., from $T[\sigma] \xrightarrow{?\mathsf{a}} T'[\sigma]$ we can infer $T \xrightarrow{?\mathsf{a}} T'$ (and therefore, by item *(i)* of Proposition 2.18, T is an external choice).

Proposition 2.19. *For all T, T' and σ,*

(i) $T \xrightarrow{?\mathsf{a}} T'$ *iff* $T[\sigma] \xrightarrow{?\mathsf{a}} T'[\sigma]$;

(ii) *for all* $\mathsf{a} \neq \mathsf{b}$, $T \xrightarrow{\tau} !\mathsf{a}.T'$ *and* $T \xrightarrow{\tau} !\mathsf{b}.T''$ *iff* $T[\sigma] \xrightarrow{\tau} T'[\sigma.!\mathsf{a}] \wedge T[\sigma] \xrightarrow{\tau} T''[\sigma.!\mathsf{b}]$;

(iii) $T \xrightarrow{!\mathsf{a}} T'$ *iff* $T[\sigma] \xrightarrow{\tau} T'[\sigma.!\mathsf{a}] \wedge \nexists \mathsf{b} \neq \mathsf{a}\,.\, T[\sigma] \xrightarrow{\tau} T'[\sigma.!\mathsf{b}]$.

Proof. See page 115. □

Proposition 2.20 below says that, up-to isomorphism, asynchronous session behaviours are *not* more general than synchronous ones, and *vice versa*: e.g., considering the session types in Example 2.17, we have $T_1 \notin \mathbb{U}_{\text{aST}}$ (i.e., there is no element of $\mathbb{U}_{\text{aST}}$ which is isomorphic to T_1), while $T_1[] \notin \mathbb{U}_{\text{ST}}$.

Proposition 2.20. *Up-to isomorphism,* $\mathbb{U}_{aST} \not\subseteq\not\supseteq \mathbb{U}_{ST}$*.*

Proposition 2.21. $T[\sigma]\xrightarrow{!\mathsf{a}}$ *iff* $\exists\sigma' \,.\, \sigma = !\mathsf{a}.\sigma'$*. If* $T \overset{!\mathsf{b}}{\Rightarrow} T'$*, then* $\exists\mathsf{a} \,.\, T[\sigma] \xrightarrow{\tau} T'[\sigma.!\mathsf{b}]\xrightarrow{!\mathsf{a}}$*.*

Proof. See page 115. □

Proposition 2.22. $T[\sigma]\overset{!!\mathsf{a}}{\Longrightarrow}$ *iff* $\exists\sigma' \,.\, \sigma = !\mathsf{a}.\sigma'$*, or* $\sigma = \epsilon$ *and* $\exists T' \,.\, T \equiv !\mathsf{a}.T'$

Proof. See page 115. □

2.3 CCS

As another class of behaviours, we now present a fragment of CCS [Mil89] without delimitations and relabelling (Definition 2.23). We will give it two semantics: one *synchronous* (Definition 2.24), and one *asynchronous* (Definition 2.25).

Definition 2.23 (CCS)**.** CCS terms *have the following syntax:*

$$P, Q, R, S \quad ::= \quad \mathbf{0} \quad | \quad \ell_\tau \,.\, P \quad | \quad P + Q \quad | \quad P \,|\, Q \quad | \quad X \quad | \quad \mu_X P$$

CCS operators are standard: non-deterministic choice $+$, parallel composition $|$, recursion (assumed to be guarded).

Definition 2.24 (Sync CCS semantics)**.** *We denote with* $\mathbb{U}_{CCS}$ *the set of behaviours of the form* P*, with transitions given by the following rules (the symmetric ones for* $|$ *and* $+$ *are omitted):*

$$\frac{}{\ell_\tau \,.\, P \xrightarrow{\ell_\tau} P} \qquad \frac{P \xrightarrow{\ell_\tau} P'}{P + Q \xrightarrow{\ell_\tau} P'} \qquad \frac{P \xrightarrow{\ell_\tau} P'}{P \,|\, Q \xrightarrow{\ell_\tau} P' \,|\, Q} \qquad \frac{P[{}^{\mu_X P}/_X] \xrightarrow{\ell_\tau} P'}{\mu_X P \xrightarrow{\ell_\tau} P'}$$

The synchronous CCS semantic rules are standard. Note that the parallel operator $|$ allows interleaved execution, but *not* synchronisation (which is provided by the LTS-level operator $\|$).

Like async session behaviours, async CCS semantics use a buffer $[\sigma]$.

Definition 2.25 (Async CCS semantics)**.** *We denote with $\mathbb{U}_{aCCS}$ the set of behaviours of the form $P[\sigma]$, with transitions given by the following rules (the symmetric ones for $+$ and $|$ are omitted):*

$$\frac{\ell_\tau \in \{\tau\} \cup \mathsf{A}^?}{(\ell_\tau \,.\, P)\,[\sigma] \xrightarrow{\ell_\tau} P[\sigma]} \qquad \frac{}{(!\mathsf{a} \,.\, P)\,[\sigma] \xrightarrow{\tau} P[\sigma \,.\, !\mathsf{a}]} \qquad \frac{}{P\,[!\mathsf{a} \,.\, \sigma] \xrightarrow{!\mathsf{a}} P[\sigma]}$$

$$\frac{P[\sigma] \xrightarrow{\ell_\tau} P'[\sigma']}{(P+Q)[\sigma] \xrightarrow{\ell_\tau} P'[\sigma']} \qquad \frac{P[\sigma] \xrightarrow{\ell_\tau} P'[\sigma']}{(P \mid Q)[\sigma] \xrightarrow{\ell_\tau} (P' \mid Q)[\sigma']} \qquad \frac{P[{}^{\mu_X P}\!/X][\sigma] \xrightarrow{\ell_\tau} P'[\sigma']}{\mu_X P[\sigma] \xrightarrow{\ell_\tau} P'[\sigma']}$$

As in async session behaviours, an output $!\mathsf{a}$ is first added at the end of the buffer, and can only be consumed from its head. Note that a behaviour *cannot* consume its own buffer; furthermore, as in the synchronous CCS semantics, $|$ just allows for interleaving. Synchronization is obtained with $P[\sigma] \parallel Q[\sigma']$, i.e. using the parallel composition of LTS states: this allows P's input actions to consume Q's output buffer, and *vice versa*.

Example 2.26. *The behaviour of the async process $!\mathsf{a} \,.\, \tau[]$ is shown as p_1 in Figure 4.1.*

We now study some relations among the classes of behaviours introduced so far. Asynchronous session behaviours can be straightforwardly encoded in async CCS.

Definition 2.27. *We define an* encoding $[\![]\!]$ of session type terms into CCS terms*:*

$$[\![\bigoplus_{i\in I} !\mathsf{a}_i.T_i]\!] = \sum_{i\in I} !\mathsf{a}_i \,.\, [\![T_i]\!] \qquad [\![\&_{i\in I} ?\mathsf{a}_i.T_i]\!] = \sum_{i\in I} ?\mathsf{a}_i \,.\, [\![T_i]\!]$$

$$[\![\mathrm{rec}_X\, T]\!] = \mu_X [\![T]\!] \qquad [\![X]\!] = X$$

By Proposition 2.28, an *async* session type and its encoding in async CCS are isomorphic.

Proposition 2.28. $T[] \cong [\![T]\!][]$.

Proposition 2.29 states that async CCS behaviours strictly include async session behaviours, while they do not include synchronous CCS, and synchronous session behaviours. Note that the encoding of (synchronous) session types in our CCS fragment is akin to the encoding of τ-less CCS into full CCS [DH87].

Proposition 2.29. *Up-to isomorphism, we have* $\mathbb{U}_{aST} \subsetneq \mathbb{U}_{aCCS} \not\supseteq \mathbb{U}_{CCS} \supsetneq \mathbb{U}_{ST}$*, and* $\mathbb{U}_{aCCS} \not\subseteq \not\supseteq \mathbb{U}_{ST}$ *and* $\mathbb{U}_{CCS} \not\subseteq \mathbb{U}_{aST}$.

Example 2.30. *The session type* $\mathsf{!a} \oplus \mathsf{!b}$ *from* $\mathbb{U}_{ST}$ *is encoded in* $\mathbb{U}_{CCS}$ *as the isomorphic behaviour* $\tau\,.\,\mathsf{!a} + \tau\,.\,\mathsf{!b}$*, while* $\mathsf{?a} \,\&\, \mathsf{?b}$ *is encoded as* $\mathsf{?a} + \mathsf{?b}$*. In the asynchronous case, by Definition 2.27, we have that the session type* $\mathsf{!a} \oplus \mathsf{!b}[\sigma]$ *from* $\mathbb{U}_{aST}$ *is encoded in* $\mathbb{U}_{aCCS}$ *as the isomorphic behaviour* $\mathsf{!a} + \mathsf{!b}[\sigma]$*, while* $\mathsf{?a} \,\&\, \mathsf{?b}[\sigma]$ *is encoded as* $\mathsf{?a} + \mathsf{?b}[\sigma]$.

Instead, $\mathsf{?a} + \mathsf{!b}$ *from* $\mathbb{U}_{CCS}$ *is* not *isomorphic to any session type in* $\mathbb{U}_{ST}$ *(due to the mixed choice between an input and an output), nor to any behaviour in* $\mathbb{U}_{aCCS}$ *(because buffered semantics would introduce a* τ*-transition on the* $\mathsf{!b}$*-branch). Moreover,* $\mathsf{?a} + \tau$ *from* $\mathbb{U}_{CCS}$ *and* $\mathsf{?a} + \tau[]$ *from* $\mathbb{U}_{aCCS}$ *are not isomorphic to any asynchronous session type in* $\mathbb{U}_{ST}$*, nor in* $\mathbb{U}_{aST}$ *(due to the choice between an input and an internal move without continuation).*

Finally, $\mathsf{!a.?b}$ *from* $\mathbb{U}_{ST}$ *is not isomorphic to any behaviour in* $\mathbb{U}_{aCCS}$ *(because the buffered semantics of the latter do not allow the* $\mathsf{!a}$*-transition to preempt the* $\mathsf{?b}$*-transition).*

Chapter 3

A motivating use case

We now introduce a running example, which will be used throughout the thesis. The following types describe the behaviours of a bartender (B), and of a customer named Alice (A):

$$\begin{aligned} U_{\mathsf{B}} &= \mathrm{rec}_X \,(?\mathsf{aCoffee}.!\mathsf{coffee}.X \;\;\&\;\; ?\mathsf{aBeer}.(!\mathsf{beer}.X \oplus !\mathsf{no}.X) \;\;\&\;\; ?\mathsf{pay}) \\ T_{\mathsf{A}} &= !\mathsf{aCoffee}.?\mathsf{coffee}.!\mathsf{pay} \;\oplus\; !\mathsf{aBeer}.(?\mathsf{beer}.!\mathsf{pay} \,\&\, ?\mathsf{no}.!\mathsf{pay}) \end{aligned}$$

The bartender presents an external choice $\&$, allowing a customer to order either coffee or beer, or to eventually pay; in the first case, he will serve the coffee and then recursively wait for more orders; in the second case, he uses the internal choice $\oplus$ to decide whether to serve the beer or not — and then waits for more orders; in the third case, after the due amount (possibly 0) is paid, the interaction ends.

Alice internally chooses between coffee or beer; in the first case, she waits to get the coffee and then pays; in the second case, she lets the bartender choose between serving the beer, or saying no — and in both cases, she will check out.

Intuitively, T_{A} and U_{B} are *compliant* — i.e., their parallel composition $T_{\mathsf{A}} \parallel U_{\mathsf{B}}$ (under Definitions 2.4 and 2.15) gives rise to a "correct" interaction such that:

1. each output of T_{A} is matched by an input of U_{B} (and *vice versa*), and
2. when they stop synchronising (after pay), they are both "successful" — i.e., they reach a final configuration $\mathbf{0} \parallel \mathbf{0}$ without further enabled transitions.

Moreover, the following processes type-check — roughly, because the pairs $U_{\mathsf{B}}, Q_{\mathsf{B}}$ and $T_{\mathsf{A}}, P_{\mathsf{A}}$ expose the same interaction labels, and have a similar branching structure:

$$\begin{aligned} Q_{\mathsf{B}} &= \mu_Y(?\mathsf{aCoffee} \,.\, !\mathsf{coffee} \,.\, Y \;+\; ?\mathsf{aBeer} \;.\; (!\mathsf{beer} \,.\, Y + !\mathsf{no} \,.\, Y) \;+\; ?\mathsf{pay}) \\ P_{\mathsf{A}} &= !\mathsf{aCoffee} \,.\, ?\mathsf{coffee} \,.\, !\mathsf{pay} \;+\; !\mathsf{aBeer} \;.\; (?\mathsf{beer} \,.\, !\mathsf{pay} + ?\mathsf{no} \,.\, !\mathsf{pay}) \end{aligned}$$

From typing and compliance, we can deduce that $P_{\mathsf{A}}[] \parallel Q_{\mathsf{B}}[]$ synchronise "correctly" (reflecting the "correct" interaction of $T_{\mathsf{A}} \parallel U_{\mathsf{B}}$), and reach the successful configuration $\mathbf{0}[] \parallel \mathbf{0}[]$, where Alice and the bartender agree in stopping their interaction.

Alice may also implement a *subtype* of T_{A} only asking for coffee: $T'_{\mathsf{A}} = !\mathsf{aCoffee}.?\mathsf{coffee}.!\mathsf{pay}$, with a corresponding process $P'_{\mathsf{A}} = !\mathsf{aCoffee} \,.\, ?\mathsf{coffee} \,.\, !\mathsf{pay}$: also in this case, we would have that T'_{A} and U_{B} are compliant, and thus $P'_{\mathsf{A}}[] \parallel Q_{\mathsf{B}}[]$ also gives rise to a "correct" interaction.

So far, the structure of A's and B's processes have matched the structure of the respective types. This is a common situation in the session types literature: processes are usually written using calculi inheriting the structured communication approach pioneered by Honda *et al.* [Hon93; HVK98], thus reflecting the internal/external choices of types. However, in some cases things may be more complex. The bartender might have other incumbencies, and he may need to stop selling beer after a certain hour:

$$\begin{aligned} Q''_{\mathsf{B}} &= \mu_Y\big((?\mathsf{aCoffee} \,.\, !\mathsf{coffee} \,.\, Y \;+\; ?\mathsf{aBeer} \;.\; (!\mathsf{beer} \,.\, Y + !\mathsf{no} \,.\, Y) \;+\; ?\mathsf{pay}) \\ &\qquad + \tau \,.\, \mu_Z(?\mathsf{aCoffee} \,.\, !\mathsf{coffee} \,.\, Z \;+\; ?\mathsf{aBeer} \,.\, !\mathsf{no} \,.\, Z \;+\; ?\mathsf{pay})\big) \end{aligned}$$

This reminds us of the small Erlang code sample given in Section 1.1.1: the τ branch represents the bartender's decision to stop waiting for customer orders, perform some internal duties (e.g. clean up the bar) and then serve again — this time, refusing to sell beer. Intuitively, we would like Q''_{B} to still have the type U_{B}, since compliant customer processes (e.g. Alice's one) will still be able to interact (either before or after the τ). A process like Q''_{B}, however, cannot usually be written (and typed) using classical session calculi: their grammar does not offer a τ prefix, since it would allow for processes where the distinction between internal/external choices is blurred (contrary to the expected program structure).

Let us consider another scenario: Alice is late for work. But she realises that the bartender-customer system is *asynchronous*: the counter is a bidirectional *buffer* where drinks and money can be placed. Thus, she tries to save time by implementing the following type and process:

$$T''_{\mathsf{A}} \;=\; \mathsf{!aCoffee}.\mathsf{!pay}.\mathsf{?coffee} \qquad\qquad P''_{\mathsf{A}} \;=\; \mathsf{!aCoffee}\,.\,(\mathsf{?coffee} \mid \mathsf{!pay})$$

i.e., in her type she plans to order a coffee, put her money on the counter while the bartender prepares her drink, and take it as soon as it is ready; in her process, she orders a coffee, and tries to grab the coffee with one hand, while putting the money on the counter with the other. P''_{A} represents an optimised program exploiting buffered communication semantics, thus diverging from the syntactic structure of T''_{A}, and reminds us of the issues discussed in Section 1.1.3. Therefore, is T''_{A} a type for P''_{A}? Is T''_{A} compliant with U_{B}, and will P''_{A} interact smoothly with Q_{B} and Q''_{B}? We shall answer these questions later on in Chapter 7.

Chapter 4

I/O compliance

We now address the problem of defining a relation between behaviours to guarantee that, when combined together, they interact in a "correct" manner. Many different notions of "correct interaction" have been considered to this purpose in the literature, on different languages and formalisms, both for the binary [Car+06; CGP09; BL10; BL14] and for the multi-party settings [BZ07b; BZ08; BCZ13; DY13]. As discussed in Section 1.1.4, we aim for a semantic, language-independent relation that fits in the framework introduced in Chapter 2.

We start by considering the classical, trace-based notion of compliance of [CGP09; BL10], where correctness is interpreted as *progress* of the interaction. In Definition 4.1 we say that a behaviour p has progress with q (in symbols, $p \dashv q$) iff, whenever a τ-computation of the system $p \parallel q$ is stuck, then p has reached the final (success) state $\mathbf{0}$. Note that this notion is *asymmetric*, in the sense that p is allowed to terminate the interaction without the permission of q. This is intended to model the asymmetry between the role of a client p and that of a server q, as in [BL10].

Definition 4.1 (Progress)**.** *We write* $p \dashv q$ *iff* $p \parallel q \Rightarrow p' \parallel q' \overset{\tau}{\not\rightarrow}$ *implies* $p' \cong \mathbf{0}$.

We write $p \dashv\vdash q$ *when* $p \dashv q$ *and* $p \vdash q$.

Example 4.2. *We have the following relations:*

$$\begin{array}{rcl@{\qquad}rcl}
\mathsf{!a}.\mathsf{?b} & \dashv\vdash & \mathsf{?a}.\mathsf{!b} & \mathrm{rec}_X\, \mathsf{!a}.X & \dashv\vdash & \mathrm{rec}_Y\, \mathsf{?a}.Y \\
\mathsf{!a}.\mathsf{?b} & \not\dashv\nvdash & \mathsf{!b}.\mathsf{?a} & (\mathsf{!a}.\mathsf{?b})\,[] & \dashv\vdash & (\mathsf{!b}.\mathsf{?a})\,[] \\
\mathsf{!a}.\mathsf{!b} & \not\dashv\nvdash & \mathsf{?c} & (\mathsf{!a}.\mathsf{!b})\,[] & \dashv\nvdash & \mathsf{?c}\,[] \\
\mathsf{!a}.\mathsf{?b} & \not\dashv\vdash & \mathsf{?a} & (\mathrm{rec}_X\, \mathsf{?a}.X)\,[] & \not\dashv\nvdash & \mathsf{!b}\,[] \\
\mathrm{rec}_X\, \mathsf{!a}.X & \not\dashv\vdash & \mathsf{?a} & (\mathrm{rec}_X\, \mathsf{!a}.X)\,[] & \dashv\vdash & \mathsf{?b}\,[]
\end{array}$$

The following proposition is a standard result on synchronous session types [BL10]: given two session types which enjoy progress, it allows to reason by cases on their structure.

Proposition 4.3 ($\dashv$/$\dashv\vdash$-induced shapes of session types)**.** *If $T \dashv U$, then exactly one of the following cases holds:*

a. $T = \mathbf{0}$;

b. $T \equiv \&_{i\in I} \mathsf{?a_i}.T_i$ *and* $U \equiv \bigoplus_{j\in J} \mathsf{!a_j}.U_j$, *with* $\emptyset \neq J \subseteq I$, *and* $\forall j \in J\,.\, T_j \dashv U_j$;

c. $T \equiv \bigoplus_{i\in I} \mathsf{!a_i}.T_i$ *and* $U \equiv \&_{j\in J} \mathsf{?a_j}.U_j$, *with* $\emptyset \neq I \subseteq J$, *and* $\forall i \in I\,.\, T_i \dashv U_i$.

Instead, if $T \dashv\vdash U$, we have either:

a. $T = U = \mathbf{0}$;

b. $T \equiv \&_{i\in I} \mathsf{?a_i}.T_i$ *and* $U \equiv \bigoplus_{j\in J} \mathsf{!a_j}.U_j$, *with* $\emptyset \neq J \subseteq I$, *and* $\forall j \in J\,.\, T_j \dashv\vdash U_j$;

c. $T \equiv \bigoplus_{i\in I} \mathsf{!a_i}.T_i$ *and* $U \equiv \&_{j\in J} \mathsf{?a_j}.U_j$, *with* $\emptyset \neq I \subseteq J$, *and* $\forall i \in I\,.\, T_i \dashv\vdash U_i$.

Proof. See [BL10]: the statement for $\dashv$ derives from the proof of Proposition 2.9 therein, while the statement for $\dashv\vdash$ derives from Lemma 3.3 in the same work (modulo two minor differences: in this paper, we postulate that $\mathbf{0}$ is the success state, and the unfolding of $\mathrm{rec}_X \cdots$ does not emit a τ-transition). □

The progress-based notion of correctness in Definition 4.1 also relates behaviours that allow apparently incorrect interactions. For instance, $(\mathrm{rec}_X\, \mathsf{!a}.X)\,[] \dashv \mathsf{?b}\,[]$ holds, because the interaction of the two behaviours produces an infinite τ-trace, even if no synchronisation ever happens. Ideally, we would like our notion of correct interaction to be stricter, avoiding "vacuous" progress where the client p exposes I/O capabilities, but the server

q cannot interact, and the composition $p \parallel q$ merely advances via internal τ-transitions (without synchronisations). We introduce a notion of compliance enjoying such a property on general behaviours (recall from Chapter 2 that $p\Downarrow^! = (p\Downarrow)^! = p\Downarrow \cap \mathsf{A}^!$).

Definition 4.4 (I/O compliance)**.** *We say that a relation $\mathcal{R}$ between behaviours is an* I/O compliance relation *iff, whenever $p \mathrel{\mathcal{R}} q$:*

a. $p\Downarrow^! \subseteq \mathrm{co}(q\Downarrow^?) \;\wedge\; \big((p\Downarrow^! = \emptyset \wedge p\Downarrow^? \neq \emptyset) \implies \emptyset \neq q\Downarrow^! \subseteq \mathrm{co}(p\Downarrow^?)\big)$*;*

b. $p \xrightarrow{\ell} p' \wedge q \xrightarrow{\mathrm{co}(\ell)} q' \implies p' \mathrel{\mathcal{R}} q'$*;*

c. $p \xrightarrow{\tau} p' \implies p' \mathrel{\mathcal{R}} q$*;*

d. $q \xrightarrow{\tau} q' \implies p \mathrel{\mathcal{R}} q'$*.*

We write $\ddot{\lhd}$ for the largest I/O compliance relation, and $\ddot{\bowtie}$ for the largest symmetric I/O compliance relation. When $p \mathrel{\ddot{\bowtie}} q$, then we say that p and q are I/O compliant*.*

Definition 4.4 can be interpreted with the game-theoretic metaphor. Let p and q be two players/behaviours. Item *a.* has two conditions: by the leftmost constraint, if p wants to do some output (possibly after some τ-moves), then q must match it with its inputs; by the rightmost constraint, if p is *not* going to output, but she wants to do some input, then q must be ready (possibly after some τ-moves) to do some output, and q cannot have outputs other than those accepted by p. I/O compliance must be preserved if p and q synchronise or do internal moves (items *b.*, *c.*, *d.*).

Item *a.* of Definition 4.4 guarantees that, in each state, the two behaviours will either stop interacting, or will keep the possibility to synchronise in the future (possibly after some τ-moves). It captures a basic duality between inputs and outputs: in correct interactions, each possible output must be matched by a corresponding input, while the *vice versa* is not necessary. In the symmetric case, item *a.* gives the intuition of underlying constraints similar to the following:

$$p\Downarrow^! \subseteq \mathrm{co}(q\Downarrow^?) \qquad\qquad q\Downarrow^! \subseteq \mathrm{co}(p\Downarrow^?)$$

giving the idea that $p \mathrel{\ddot{\lhd}} q$ is *covariant* w.r.t. p's outputs and q's inputs, and *contravariant* w.r.t. p's inputs and q's outputs. Indeed, both constraints above are explicit in the *symmetric* relation $\ddot{\bowtie}$, which forfeits client/server distinctions.

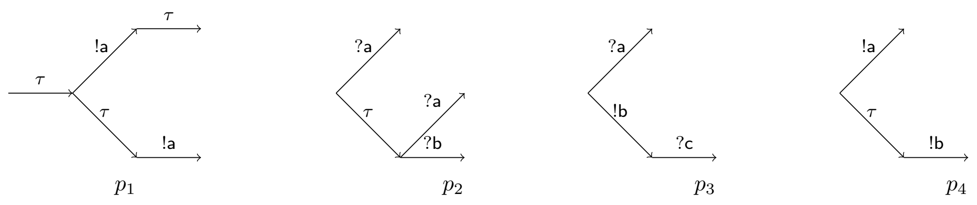


Figure 4.1: Four behaviours which are not session behaviours.

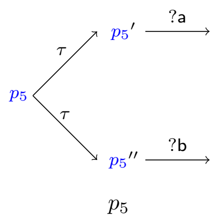


Figure 4.2: A behaviour which does not admit an I/O compliant one.

Lemma 4.5 below says that the largest symmetric I/O compliance relation $\ddot{\bowtie}$ coincides with the intersection of $\ddot{\lhd}$ and its inverse.

Lemma 4.5. $\ddot{\bowtie} \;=\; \ddot{\rhd} \cap \ddot{\lhd}$.

Proof. See page 118. □

Example 4.6. *Consider the behaviours in Figure 4.1. We have that* $p_1 \mathrel{\ddot{\bowtie}} p_2$, $p_2 \mathrel{\ddot{\bowtie}} p_4$, $p_1 \mathrel{\ddot{\lhd}} p_3$, *and* $p_2 \mathrel{\ddot{\lhd}} p_3$, *while all the other pairs of behaviours are not compliant.*

The following example shows that there exist behaviours which do not admit a (symmetric) I/O compliant one.

Example 4.7. *Consider the behaviour* p_5 *in Figure 4.2. We show that, for all behaviours* q, *it must be* $q \mathrel{\not\ddot{\bowtie}} p_5$. *By contradiction, assume that* $q \mathrel{\ddot{\bowtie}} p_5$, *for some* q. *Since* $p_5 \xrightarrow{\tau} p_5'$ *and* $p_5 \xrightarrow{\tau} p_5''$, *by applying twice clause d. of Definition 4.4 it follows that* $q \mathrel{\ddot{\bowtie}} p_5'$ *and* $q \mathrel{\ddot{\bowtie}} p_5''$. *Then, by applying twice the first part of clause a., we have:*

$$q{\Downarrow}^{!} \;\subseteq\; \mathrm{co}(p_5'{\Downarrow}^{?}) \cap \mathrm{co}(p_5''{\Downarrow}^{?}) \;=\; \{?\mathsf{a}\} \cap \{?\mathsf{b}\} \;=\; \emptyset$$

Now, since $p_5' \mathrel{\ddot{\bowtie}} q$ *and* $p_5'{\Downarrow}^{!} = \emptyset \neq p_5'{\Downarrow}^{?}$, *by the second part of clause a. we would obtain* $\emptyset \neq q{\Downarrow}^{!}$ *— contradiction.*

We now relate I/O compliance with the notion of progress introduced in Definition 4.1. Item (a) of Theorem 4.9 below states that, if two behaviours are compliant, then they enjoy progress. The *vice versa* is not always true, as shown in Example 4.8.

Example 4.8. *We have* $(\mathrm{rec}_X\,!\mathsf{a}.X)[\,] \dashv ?\mathsf{b}[\,]$*, but* $(\mathrm{rec}_X\,!\mathsf{a}.X)[\,] \not\ddot{\lhd}\, ?\mathsf{b}[\,]$*: in fact, the weak output barbs of the LHS contain* $!\mathsf{a}$*, which is not matched by the weak input barbs of the RHS (that only contain* $?\mathsf{b}$*) — thus violating item a. of Definition 4.4. This is coherent with our* desideratum *that correct interactions must not progress vacuously.*

The fact that $\dashv\, \not\subseteq\, \ddot{\lhd}$ *can also be seen on synchronous CCS. Consider, for instance,* $P = !\mathsf{a} + !\mathsf{b}$ *and* $Q = ?\mathsf{a}$*: we have* $P \dashv Q$ *(because the LHS and RHS can synchronise on* a*, and then terminate), but* $P \not\ddot{\lhd}\, Q$ *(because* $!\mathsf{b}$ *on the LHS is not matched by the inputs of the RHS).*

Still, $\ddot{\lhd}$ and $\dashv$ coincide for *synchronous* session behaviours, as shown in item (b)) of Theorem 4.9. The intuition is the following. Take $T, U \in \mathbb{U}_{\mathrm{ST}}$: by Definition 2.15, they cannot generate infinite τ-traces, and whenever they output, they always commit to a *single* output transition. These properties allow to avoid situations similar to the ones in Example 4.8; as a consequence, if T, U enjoy progress, then they are also I/O compliant.

Theorem 4.9. *(a) If* $p \mathrel{\ddot{\lhd}} q$*, then* $p \dashv q$*; (b) if* $p, q \in \mathbb{U}_{ST}$ *and* $p \dashv q$*, then* $p \mathrel{\ddot{\lhd}} q$*.*

Proof. We first prove $\ddot{\lhd} \subseteq \dashv$. Let:

$$F(\mathcal{X}) \quad = \quad \left\{ (p,q) \;\middle|\; \begin{array}{ll} p \parallel q \not\xrightarrow{\tau} \text{ implies } p \cong \mathbf{0} & \text{and} \\ p \parallel q \xrightarrow{\tau} p' \parallel q' \text{ implies } (p',q') \in \mathcal{X} & \end{array} \right\}$$

It is easy to check that $\mathrm{gfp}(F) = \dashv$. By the coinduction proof principle, if $\ddot{\lhd} \subseteq F(\ddot{\lhd})$, then we can deduce that $\ddot{\lhd} \subseteq \mathrm{gfp}(F) = \dashv$. So, we show that $\ddot{\lhd} \subseteq F(\ddot{\lhd})$. Let $p \mathrel{\ddot{\lhd}} q$.

For the first clause of the definition of F, assume $p \parallel q \not\xrightarrow{\tau}$. We show that p has no outgoing transitions (i.e., $p \cong \mathbf{0}$) by showing the absurdity of the following cases:

- $p \xrightarrow{\tau}$. Not possible, because it would contradict the assumption $p \parallel q \not\xrightarrow{\tau}$.
- $p \xrightarrow{!\mathsf{a}}$. By item a. of Definition 4.4, we have that $p{\Downarrow}^{!} \subseteq \mathrm{co}(q{\Downarrow}^{?})$, hence $q \overset{?\mathsf{a}}{\Longrightarrow}$. Note that $q \not\xrightarrow{\tau}$, because otherwise we would contradict the assumption $p \parallel q \not\xrightarrow{\tau}$. Therefore, it must be $q \xrightarrow{?\mathsf{a}}$, and so by definition of $\parallel$ we would obtain $p \parallel q \xrightarrow{\tau}$ — contradiction.
- $p \xrightarrow{?\mathsf{a}}$ and $\nexists \mathsf{b}\,.\, p \xrightarrow{!\mathsf{b}}$. Note that $p \not\xrightarrow{\tau}$ and $q \not\xrightarrow{\tau}$, because otherwise we would contradict the assumption $p \parallel q \not\xrightarrow{\tau}$. Since $p{\Downarrow}^{!} = \emptyset$ and $p{\Downarrow}^{?} \neq \emptyset$, by item a. of Definition 4.4 we have that $\emptyset \neq q{\Downarrow}^{!} \subseteq \mathrm{co}(p{\Downarrow}^{?})$. Since $q \not\xrightarrow{\tau}$, it must be $q \xrightarrow{!\mathsf{a}}$, and so by definition of $\parallel$ we would obtain $p \parallel q \xrightarrow{\tau}$ — contradiction.

Thus, since p cannot have outgoing transitions, we conclude that $p \cong \mathbf{0}$.

For the second clause of the definition of F, assume that $p \parallel q \xrightarrow{\tau}$. We have the following three cases:

- $p \xrightarrow{\tau} p'$. By definition of $\parallel$, this implies $p \parallel q \xrightarrow{\tau} p' \parallel q$. By item *c.* of Definition 4.4, we conclude that $p' \mathrel{\ddot{\lhd}} q$.
- $q \xrightarrow{\tau} q'$. Similar to the previous case.
- $p \xrightarrow{\ell} p'$ and $q \xrightarrow{\text{co}(\ell)} q'$, for some label ℓ. By definition of $\parallel$, this implies $p \parallel q \xrightarrow{\tau} p' \parallel q'$. By item *b.* of Definition 4.4, we conclude that $p' \mathrel{\ddot{\lhd}} q'$.

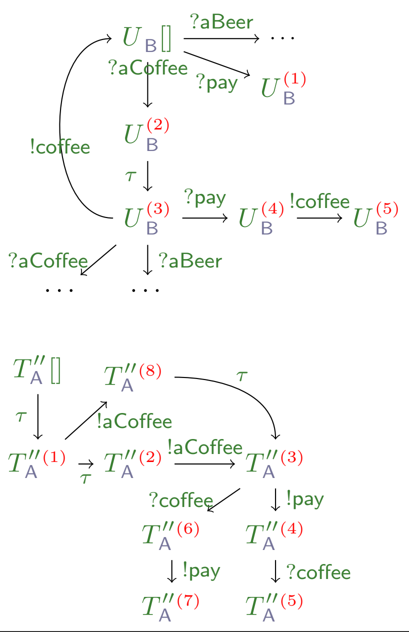
In all three cases, we have concluded that $p' \mathrel{\ddot{\lhd}} q'$, thus satisfying the second clause of F.

We now prove that $T \dashv U$ implies $T \mathrel{\ddot{\lhd}} U$, for all session types T, U. To do that, it suffices to show that $\dashv$ is an I/O compliance relation. Assume that $T \dashv U$. We show that all the clauses of Definition 4.4 are satisfied:

- recall the 3 possible forms of T and U from Proposition 4.3: modulo unfolding, either $T = \mathbf{0}$, or $T = \bigoplus_{i \in I} !\mathsf{a}_i . T_i$ (with $|I| \geq 1$) and $U = \&_{j \in J} ?\mathsf{a}_j . U_j$ with $I \subseteq J$, or *vice versa* (i.e., T is an external choice and U is an internal choice). Each of these three possible forms satisfies item *a.* in Definition 4.4;
- when the premise of item *b.* of Definition 4.4 holds, we have $T \parallel U \xrightarrow{\tau} T' \parallel U'$; by definition of progress, we have $T' \dashv U'$.
- when $T \xrightarrow{\tau} T'$ (premise of item *c.* of Definition 4.4), by definition of progress, we have $T' \dashv U$.
- similarly, when $U \xrightarrow{\tau} U'$ (premise of item *d.* of Definition 4.4), by definition of progress, we have $T \dashv U'$. □

Note that I/O compliance can relate asynchronous session behaviours which intuitively interact correctly, but which would *not* be compliant under the synchronous semantics, e.g. $(!\mathsf{a}.?\mathsf{b})\,[] \mathrel{\ddot{\bowtie}} (!\mathsf{b}.?\mathsf{a})\,[]$.

Example 4.10. *Recall the types and processes in Chapter 3. In the sync case,* $U_{\mathsf{B}} \dashv\vdash T_{\mathsf{A}}$, $U_{\mathsf{B}} \mathrel{\ddot{\bowtie}} T_{\mathsf{A}}$, $U_{\mathsf{B}} \dashv\vdash T'_{\mathsf{A}}$ *and* $U_{\mathsf{B}} \mathrel{\ddot{\bowtie}} T'_{\mathsf{A}}$. *The same holds for their async versions.*



Relation
$(U_{\mathsf{B}}[], T''_{\mathsf{A}}[])$
$(U_{\mathsf{B}}[], T''^{(1)}_{\mathsf{A}})$
$(U_{\mathsf{B}}[], T''^{(2)}_{\mathsf{A}})$
$(U_{\mathsf{B}}^{(2)}, T''^{(8)}_{\mathsf{A}})$
$(U_{\mathsf{B}}^{(2)}, T''^{(3)}_{\mathsf{A}})$
$(U_{\mathsf{B}}^{(3)}, T''^{(8)}_{\mathsf{A}})$
$(U_{\mathsf{B}}^{(3)}, T''^{(3)}_{\mathsf{A}})$
$(U_{\mathsf{B}}^{(4)}, T''^{(4)}_{\mathsf{A}})$
$(U_{\mathsf{B}}^{(5)}, T''^{(5)}_{\mathsf{A}})$
$(U_{\mathsf{B}}[], T''^{(6)}_{\mathsf{A}})$
$(U_{\mathsf{B}}^{(1)}, T''^{(7)}_{\mathsf{A}})$

Note that $U_{\mathsf{B}}[]$ has an input (?exAskBeer) which is not matched by an output of $T''_{\mathsf{A}}[]$, and therefore leads to states not considered in the relation (and here omitted). The same goes for $U_{\mathsf{B}}^{(3)}$, with two unmatched inputs (?aBeer and ?aCoffee).
The states in the transition diagram of $T''_{\mathsf{A}}[]$ are detailed in Table 6.1.

Table 4.1: Example of I/O compliance.

When Alice is late for work, for the synchronous *types we have* $U_{\mathsf{B}} \not\dashv\vdash T''_{\mathsf{A}}$ *and* $U_{\mathsf{B}} \not\ddot{\bowtie} T''_{\mathsf{A}}$*, due to the wrong order of Alice's actions. In the* asynchronous *case, instead, we have* $U_{\mathsf{B}}[] \dashv\vdash T''_{\mathsf{A}}[]$ *and* $U_{\mathsf{B}}[] \ddot{\bowtie} T''_{\mathsf{A}}[]$*. The latter relation is detailed in Table 4.1.*

The following lemma characterises the form of buffers in asynchronous computations of synchronously-compliant session behaviours.

Lemma 4.11 (Half-duplex communication in compliant async session behaviours)**.** *Let* $T \circ U$*, for some* $\circ \in \{\dashv, \vdash, \dashv\vdash, \ddot{\lhd}, \ddot{\rhd}, \ddot{\bowtie}\}$*, and assume that* $T[] \parallel U[] \Rightarrow T'[\sigma] \parallel U'[\rho]$*. Then:*

(i) $\sigma = \epsilon$ *or* $\rho = \epsilon$.

(ii) *if* $\sigma = \rho = \epsilon$ *then* $T' \circ U'$.

Proof. See page 121. □

Proposition 4.12 states that for session types, I/O compliance under the synchronous semantics is preserved when passing to the asynchronous semantics. As we shall see, some form of preservation when passing from the synchronous to the asynchronous semantics holds for all the main relations studied in this thesis.

Proposition 4.12. *If* $T \ddot{\lhd} U$*, then* $T[] \ddot{\lhd} U[]$*.*

Proof. Assume that $T \mathrel{\ddot{\lhd}} U$, and let:

$$\mathcal{R} = \left\{(T'[\sigma], U'[\rho]) \;\middle|\; T[] \parallel U[] \Rightarrow T'[\sigma] \parallel U'[\rho]\right\}$$

We will prove that $\mathcal{R}$ is an I/O compliance relation, i.e. it satisfies all the items of Definition 4.4. The thesis will then follow by the fact that $(T[], U[]) \in \mathcal{R}$.

Let $(T'[\sigma], U'[\rho]) \in \mathcal{R}$. Note that items *b.–d.* of Definition 4.4 follow directly by definition of $\parallel$, so we only need to prove item *a.*. By Lemma 4.11, we have $\sigma = \epsilon$ or $\rho = \epsilon$, and so we have the following three cases:

- $\sigma \neq \epsilon$ and $\rho = \epsilon$. Let $\sigma = !\mathsf{a}.\sigma'$. Then:

$$T'[\sigma]\Downarrow^{!} = \{!\mathsf{a}\} \tag{4.1}$$

  By items *(ii)* and *(iii)* of Proposition 2.19 we know that each output in σ has been generated by some T'' such that $T'' \overset{\sigma}{\Rightarrow} T'$; let us take such T'' so that:

$$T[] \parallel U[] \;\Rightarrow\; T''[] \parallel U'[] \;\Rightarrow\; T'[\sigma] \parallel U'[]$$

  Since $T \mathrel{\ddot{\lhd}} U$ and $T[] \parallel U[] \Rightarrow T''[] \parallel U'[]$, from Lemma 4.11 we obtain that $T'' \mathrel{\ddot{\lhd}} U'$. By items *(ii)* and *(iii)* of Proposition 2.18, we know that T'' is an internal choice, hence by Theorem 4.9 and Proposition 4.3 we know that U' must be a (larger) external choice, i.e. $T''\Downarrow^{!} \subseteq \mathrm{co}(U'\Downarrow^{?})$. Hence, from Equation (4.1) we obtain:

$$\{!\mathsf{a}\} \;=\; T'[\sigma]\Downarrow^{!} \;\subseteq\; T''\Downarrow^{!} \;\subseteq\; \mathrm{co}(U'\Downarrow^{?}) \;=\; \mathrm{co}(U'[]\Downarrow^{?})$$

  This proves the first part of item *a.* in Definition 4.4. The second part holds vacuously, because $T'[\sigma]\Downarrow^{!} \neq \emptyset$.

- $\sigma = \epsilon$ and $\rho \neq \epsilon$. If $T' = \mathbf{0}$, then item *a.* in Definition 4.4 is trivially satisfied. Otherwise, the reasoning is similar to the case above, by swapping the role of T' and U', and considering the queue ρ instead of σ. More in detail, we let $\rho = !\mathsf{a}.\rho'$, and so we obtain:

$$U'[\rho]\Downarrow^{!} = \{!\mathsf{a}\} \tag{4.2}$$

As in the previous item, via Proposition 2.19 we take U'' such that $U'' \overset{\rho}{\Rrightarrow} U'$, from which we have:

$$T[\,] \parallel U[\,] \;\Rightarrow\; T'[\,] \parallel U''[\,] \;\Rightarrow\; T'[\,] \parallel U'[\rho]$$

Since $T \ddot{\lhd} U$ and $T[\,] \parallel U[\,] \Rightarrow T'[\,] \parallel U''[\,]$, from Lemma 4.11 we obtain that $T' \ddot{\lhd} U''$. By Proposition 2.18 we have that U'' is a non-empty internal choice, and by Theorem 4.9 and Proposition 4.3 we know that T' is a (larger) external choice, i.e. $\emptyset \neq U''\Downarrow^{!} \subseteq \mathrm{co}(T'\Downarrow^{?})$. Since T' is an external choice, then $T'[\,]\Downarrow^{!} = \emptyset$, which vacuously satisfies the first part of item *a.* in Definition 4.4. For the second part of the item, from Equation (4.2) we have:

$$\emptyset \;\neq\; \{!\mathsf{a}\} \;=\; U'[\rho]\Downarrow^{!} \;\subseteq\; U''\Downarrow^{!} \;\subseteq\; \mathrm{co}(T'\Downarrow^{?}) \;=\; \mathrm{co}(T'[\,]\Downarrow^{?})$$

- $\sigma = \rho = \epsilon$. By Lemma 4.11, it follows that $T' \ddot{\lhd} U'$. Hence, by Theorem 4.9 and Proposition 4.3, we can proceed by cases on the form of T' and U':
  - $T' = \mathbf{0}$. Then, item *a.* of Definition 4.4 is trivially satisfied.
  - $T' = \&_{i\in I} ?\mathsf{a}_i.T_i$ and $U' = \bigoplus_{j\in J} !\mathsf{a}_j.U_j$, with $\emptyset \neq J \subseteq I$. Then, $T'[\,]\Downarrow^{!} = \emptyset$, which satisfies the first part of item *a.*. Further, $\emptyset \neq U'[\,]\Downarrow^{!} \subseteq \mathrm{co}(T'[\,]\Downarrow^{?})$, thus satisfying the second part of the item.
  - $T' = \bigoplus_{i\in I} !\mathsf{a}_i.T_i$ and $U' = \&_{j\in J} ?\mathsf{a}_j.U_j$, with $\emptyset \neq I \subseteq J$. Then, $T'[\,]\Downarrow^{!} \subseteq \mathrm{co}(U'[\,]\Downarrow^{?})$, which satisfies the first part of item *a.*; the second part of the item is vacuously true, since $T'[\,]\Downarrow^{!} \neq \emptyset$. □

The following theorem extends Proposition 4.12 to all the notions of compliance studied in this chapter.

Theorem 4.13. *If* $T \circ U$, *then* $T[\,] \circ U[\,]$, *for* $\circ \in \{\vdash, \dashv\vdash, \dashv, \ddot{\rhd}, \ddot{\bowtie}, \ddot{\lhd}\}$.

Proof. The statement for $\circ = \ddot{\lhd}$ has already been proved in Proposition 4.12.

For $\circ = \dashv$, we have:

$$\begin{tikzcd}
T \dashv U \arrow[r, dashed] \arrow[d, Rightarrow, "\text{Theorem 4.9(b)}"'] & T[\,] \dashv U[\,] \\
T \ddot{\lhd} U \arrow[r, Rightarrow, "\text{Proposition 4.12}"] & T[\,] \ddot{\lhd} U[\,] \arrow[u, Rightarrow, "\text{Theorem 4.9(a)}"']
\end{tikzcd}$$

The proofs for $\circ \in \{\vdash, \ddot{\rhd}\}$ follow by symmetry, while the proofs for $\circ \in \{\dashv\vdash, \ddot{\bowtie}\}$ follow respectively from $\dashv\vdash \; = \; \vdash \cap \dashv$ (by Definition 4.1) and $\ddot{\bowtie} = \ddot{\rhd} \cap \ddot{\lhd}$ (by Lemma 4.5). $\square$

Chapter 5

On safety

In Chapter 4, we claimed that I/O compliance represents a "correct" notion of interaction, since it avoids "vacuous" progress (see Example 4.2). In this chapter, we study how such a notion of "correctness" is related to classical notions of *deadlock-freedom* and *safety*, with a prominent focus on asynchronous session behaviours (as introduced in Section 2.2).

5.1 Deadlock states

In this section, we introduce a semantic notion of *deadlock state*.

Definition 5.1 (Deadlock state)**.** p is a deadlock state *iff* $p\rightarrow$ *and* $p\overset{\tau}{\not\rightarrow}$.

Intuitively, p is a deadlock state if it exposes I/O interaction capabilities (i.e., $\exists \ell . p\xrightarrow{\ell}$), but cannot reduce via internal τ transitions: therefore, p requires some external interaction in order to reduce further. The meaning of such a definition becomes apparent when applied on parallel compositions of behaviours, as in Corollary 5.2 below.

Corollary 5.2. *$p \parallel q$ is a deadlock state iff* all *the following hold:*

(i) $p \not\cong \mathbf{0}$ *or* $q \not\cong \mathbf{0}$*;*

(ii) $p\overset{\tau}{\not\rightarrow}$ *and* $q\overset{\tau}{\not\rightarrow}$*;*

(iii) $\forall \ell \ . \ p\xrightarrow{\ell}$ *implies* $q\overset{\mathrm{co}(\ell)}{\not\longrightarrow}$.

Proof. For the $\implies$ direction, assume that $p \| q$ is a deadlock state: then (by Definition 5.1) we have $p \parallel q \!\rightarrow$, which implies item *(i)* above; furthermore, we have $p \parallel q \overset{\tau}{\not\rightarrow}$, which implies items *(ii)*–*(iii)* (in fact, if one of such items does *not* hold, we have the contradiction $p \parallel q \xrightarrow{\tau}$).

For the $\impliedby$ direction, note that item *(i)* implies $p \parallel q \!\rightarrow$; furthermore, items *(ii)*–*(iii)* imply $p \parallel q \overset{\tau}{\not\rightarrow}$. □

Armed with Definition 5.1 and Corollary 5.2, we can see that the notion of progress introduced in Definition 4.1 corresponds to the notion of deadlock-freedom in Definition 5.1.

Proposition 5.3 (Progress is deadlock-freedom). $p \dashv\vdash q$ *iff, for all* r, $p \parallel q \Rightarrow r$ *implies that* r *is* not *a deadlock state.*

Proof. Let $p \parallel q \Rightarrow r = p' \parallel q'$.

For the $\implies$ direction, assume $p \dashv\vdash q$: by Definition 4.1, we have either $p' \parallel q' \xrightarrow{\tau}$ (which violates Definition 5.1), or $r \cong \mathbf{0} \not\rightarrow$.

For the $\impliedby$ direction, assume $p \parallel q \Rightarrow r \cong p' \parallel q'$, with $r \overset{\tau}{\not\rightarrow}$. Since r is *not* a deadlock state, then by Definition 5.1 we have $r \not\rightarrow$. Then, $p' \cong \mathbf{0}$ and $q' \cong \mathbf{0}$. □

We can now show that I/O compliance does not hold in deadlock states (Lemma 5.4).

Lemma 5.4. *If* $p \parallel q$ *is a deadlock state, then* $p \not\ddot{\bowtie} q$.

Proof. By contradiction, assume $p \ddot{\bowtie} q$: then, by Theorem 4.9, we have $p \dashv\vdash q$, which by Proposition 5.3 implies that $p \parallel q$ is *not* a deadlock state (contradiction). □

5.2 Orphan messages

In this section, we tackle the problem of formalising the notion of *orphan message* in our LTS-based setting. Intuitively, orphan messages are outputs that are sent, but never received. They are typically studied in the setting of Communicating Finite State Machines (CFSM) [BZ83], especially when applied to the characterisation of (multiparty) session types, and to the synthesis choreographies (see e.g. [LTY15; DY13]); in the binary session types setting, orphan messages are also studied in [CDY14].

Let us consider a parallel composition of processes $p \parallel q$. Intuitively, if p, possibly after some internal moves, will always eventually expose an output transition $!\mathsf{a}$, but q (even after some internal moves) never exposes a matching input transition $?\mathsf{a}$, then we can say that $!\mathsf{a}$ is "orphan", according to Definition 5.5 below. Furthermore, by Proposition 5.6, we have that such a condition is persistent along the τ-transitions of p and q.

Definition 5.5 (Orphan message configuration)**.** *We say that* $p \parallel q$ is an orphan message configuration *iff* $\exists \mathsf{a} \,.\, p \overset{!!\mathsf{a}}{\Longrightarrow}$ *and* $q \overset{?\mathsf{a}}{\not\Longrightarrow}$ *(or* vice versa*). In this case, we say that* $p \parallel q$ *is an orphan message configuration for* p *(resp.* q*), and* $!\mathsf{a}$ is an orphan message of p (resp. q)*.*

Proposition 5.6. *Let* $p \parallel q$ *be an orphan message configuration, with* $!\mathsf{a}$ *orphan message of* p*. If* $p \Rightarrow p'$ *and* $q \Rightarrow q'$*, then* $p' \parallel q'$ *is an orphan message configuration, with* $!\mathsf{a}$ *orphan message of* p'*.*

Proof. By Definition 5.5, we have $p \overset{!!\mathsf{a}}{\Longrightarrow}$ and $q \overset{?\mathsf{a}}{\not\Longrightarrow}$. From Definition 2.6, we have $p' \overset{!!\mathsf{a}}{\Longrightarrow}$; furthermore, $q \overset{?\mathsf{a}}{\not\Longrightarrow}$ implies $q' \overset{?\mathsf{a}}{\not\Longrightarrow}$. Therefore, by Definition 5.5 we conclude that $p' \parallel q'$ is an orphan message configuration, with $!\mathsf{a}$ orphan message of p'. □

Definition 5.5 generalises the usual notion of orphan message from CFSM literature, and it is also stricter: differently from [LTY15; DY13], our definition does *not* require a processes/machines to be terminated (we will further discuss this issue in Example 5.16). This fact becomes apparent in the setting of asynchronous session types: by Proposition 2.22 below, we have that an output at the head of a buffer is persistently enabled, as required by Definition 2.6. Therefore, if we have a parallel composition $T[!\mathsf{a}.\sigma] \parallel U[\rho]$ where $!\mathsf{a}$ is an orphan message of $T[!\mathsf{a}.\sigma]$ (by Definition 5.5 above), then $!\mathsf{a}$ will remain stuck at the head of the buffer without being consumed; and furthermore, since $!\mathsf{a}$ precedes other outputs in σ, these outputs will remain unconsumed as well: although the parallel composition may perform more τ-transitions, the buffer can only grow (Proposition 5.8 below).

Lemma 5.7. $T[\sigma] \parallel U[\rho]$ *is an orphan message configuration with* $!\mathsf{a}$ *orphan message of* $T[\sigma]$ *iff both the following conditions hold:*

(i) $\sigma = !\mathsf{a}.\sigma'$, *or* $\sigma = \epsilon$ *and* $T \equiv !\mathsf{a}.T'$*;*

(ii) $U[\rho] \overset{?\mathsf{a}}{\not\Longrightarrow}$*.*

Proof. Follows directly from Definition 5.5 and Proposition 2.22. □

Proposition 5.8. *If $T[\sigma] \parallel U[\rho]$ is an orphan message configuration with $!\mathsf{a}$ orphan message of $T[\sigma]$, then $T[\sigma] \parallel U[\rho] \Rightarrow T'[\sigma'] \parallel U'[\rho']$ implies that $T'[\sigma'] \parallel U'[\rho']$ is an orphan message configuration, with $!\mathsf{a}$ orphan message of $T'[\sigma']$. Furthermore, $\sigma' = \sigma.\sigma''$ (for some σ'').*

Proof. By hypothesis and Definition 5.5 we have $T[\sigma] \overset{!!\mathsf{a}}{\Longrightarrow}$ and $U[\rho] \overset{?\mathsf{a}}{\not\Longrightarrow}$. Moreover, by Lemma 5.7 we have either $\sigma = !\mathsf{a}.\sigma''$ (for some σ''), or $\sigma = \epsilon$ and $T = !\mathsf{a}.T'$: therefore, after at most one τ-step (when $\sigma = \epsilon$), $!\mathsf{a}$ is at the head of T's buffer and becomes *only* output (weakly) reachable from $T[\sigma]$ (by Proposition 2.22). At that point, since $U[\rho] \overset{?\mathsf{a}}{\not\Longrightarrow}$, the two behaviours cannot synchronise on such $!\mathsf{a}$, and thus it cannot be removed from the head of the buffer. Hence, each τ-move along the trace $T[\sigma] \parallel U[\rho] \Rightarrow T'[\sigma'] \parallel U'[\rho']$ may be generated in only two ways, by Definition 2.16:

a. T or U add new outputs to their respective buffers;

b. T reaches an external choice which synchronises with the head of U's queue, consuming it.

(Note that case *b.* prevents us from resorting to Proposition 5.6, since the latter only considers the *internal* τ-transitions of the behaviours, without synchronisations). Therefore, $\sigma' = \sigma.\sigma''$ for some σ'' deriving from case *a.* above (which proves the *"furthermore..."* part of the statement), and $U'[\rho'] \overset{?\mathsf{a}}{\not\Longrightarrow}$: by Definition 5.5, we conclude that $T'[\sigma'] \parallel U'[\rho']$ is still an orphan message configuration, with $!\mathsf{a}$ orphan message of $T'[\sigma']$. □

Proposition 5.8 highlights a property of asynchronous session behaviours: they preserve orphan messages *even after synchronising* (case *b.* in the proof) — unlike the general behaviours considered in Proposition 5.6.

Lemma 5.9. *If $p \parallel q$ is an orphan message configuration for p, then $p \not\ddot{\dashv} q$.*

Proof. By Definition 5.5, we have $p \overset{!!\mathsf{a}}{\Longrightarrow}$ and $q \overset{?\mathsf{a}}{\not\Longrightarrow}$: this implies $!\mathsf{a} \in p\Downarrow^{!} \not\subseteq q\Downarrow^{?}$, which violates clause *a.* of Definition 4.4. Therefore, we have $p \not\ddot{\dashv} q$. □

From Lemma 5.9 we can also easily obtain that, if $p \parallel q$ is an orphan message configuration for q, then $p \not\ddot{\vdash} q$.

Example 5.10. *The inverse implication of Lemma 5.9 does* not *hold on general behaviours. For instance, consider $P = \mathsf{!a} + \tau \,.\, \mathbf{0}$: we have $P \not\mathrel{\ddot{\lhd}} \mathbf{0}$, because $\mathsf{!a} \in P{\Downarrow}^{!} \not\subseteq \mathrm{co}(\mathbf{0}{\Downarrow}^{?}) = \emptyset$ (which violates clause a. of Definition 4.4); however, $P \,\|\, \mathbf{0}$ is* not *an orphan message configuration, because $P \overset{\mathsf{!!a}}{\not\Rightarrow}$. Similar behaviours will be reprised and further discussed later, in Example 5.23.*

5.3 Unspecified reception

Another typical notion of "unsafe interaction" from CFSM literature is that of *"unspecified reception"* [CF05]. Similarly to orphan messages, unspecified reception configurations are also studied in (multiparty) session types and choreography synthesis literature (see e.g. [LTY15; DY13]).

Intuitively, in our semantic setting, unspecified reception configurations occur when a behaviour p can only interact via input transitions, but it is composed in parallel with some q which is not going to offer any matching output. Before formalising this notion in Definition 5.15 below, we need an auxiliary definition.

Definition 5.11 (Input behaviour)**.** *We say that p* is an input behaviour *(written $p \overset{??}{\Longrightarrow}$) whenever:*

$$p{\Downarrow}^{!} = \emptyset \qquad\qquad \textit{and} \qquad\qquad p \Rightarrow p' \;\textit{ implies } p{\Downarrow}^{?} \neq \emptyset$$

Intuitively, Definition 5.11 says that an input behaviour must *not* expose outputs ($p{\Downarrow}^{!} = \emptyset$) and must expose some (weak) input transitions after any τ-move ($p'{\Downarrow}^{?} \neq \emptyset$). By Proposition 5.12 below, an input behaviour cannot stop interacting after some τ-moves; furthermore, by Proposition 5.13, an input behaviour can only τ-reduce to an input behaviour.

Proposition 5.12. *If p is an input behaviour, then $p \Rightarrow p'$ implies $p' \not\gtrsim \mathbf{0}$.*

Proof. From Definition 5.11, we have that $\exists p'', \mathsf{a} \,.\, p' \Rightarrow p'' \xrightarrow{?\mathsf{a}}$: therefore, $p'' \not\gtrsim \mathbf{0}$, which in turn implies $p' \not\gtrsim \mathbf{0}$. □

Proposition 5.13. *If $p \overset{??}{\Longrightarrow}$ and $p \Rightarrow p'$, then $p' \overset{??}{\Longrightarrow}$.*

Proof. Follows from Definition 5.11: from $p{\Downarrow^!} = \emptyset$ we have $p'{\Downarrow^!} = \emptyset$; moreover, since $\forall p'' \,.\, p \Rightarrow p''$ implies $p''{\Downarrow^?} \neq \emptyset$, and $p \Rightarrow p'$ (by hypothesis), we have $\forall p'' \,.\, p' \Rightarrow p''$ implies $p''{\Downarrow^?} \neq \emptyset$. Hence, by Definition 5.11, we conclude $p' \overset{??}{\Longrightarrow}$. □

An immediate example of input behaviour is the external choice of (synchronous) session types. Another example is the external choice of asynchronous session types with empty buffer, according to Proposition 5.13 below.

Proposition 5.14. *$T[\sigma] \overset{??}{\Longrightarrow}$ iff $T \equiv \&_{i \in I} ?\mathsf{a}_i . T_i$ with $I \neq \emptyset$, and $\sigma = \epsilon$.*

Proof. Straightforward, by Definition 2.16. □

We can now formalise the notion of unspecified reception.

Definition 5.15 (Unspecified reception configuration)**.** *We say that $p \| q$ is an* unspecified reception configuration *iff $p \overset{??}{\Longrightarrow}$ and $\nexists \mathsf{a} \,.\, p \overset{?\mathsf{a}}{\Longrightarrow} \wedge q \overset{!\mathsf{a}}{\Longrightarrow}$ (or* vice versa*). In this case, we say that $p \parallel q$ is an unspecified reception configuration for p (resp. q).*

Example 5.16. *$\mathrm{rec}_X\, !\mathsf{a}.X[] \parallel ?\mathsf{b}[]$ is an unspecified reception configuration for the RHS behaviour (and also an orphan message configuration for the LHS behaviour, by Definition 5.5). Note that, in several CFSM papers (e.g. [LTY15; DY13]), such a configuration is* not *(and never becomes) an orphan message configuration, since the LHS behaviour is not final (and never terminates).*

$\mathbf{0}[] \parallel ?\mathsf{b}[]$ is an unspecified reception configuration for the RHS behaviour (and also a deadlock state, by Definition 5.1).

$?\mathsf{a}[] \parallel ?\mathsf{b}[]$ is an unspecified reception configuration for both LHS and RHS behaviours (and also a deadlock state).

Proposition 5.17. *Let $p \parallel q$ be an unspecified reception configuration for p. Then, $p \parallel q \Rightarrow p' \parallel q'$ implies that $p' \parallel q'$ is an unspecified reception configuration for p'.*

Proof. We first notice that, by Definition 5.15, p and q cannot synchronise, because no output of q matches p's inputs, — and p has no weakly reachable outputs. Therefore, the τ transitions along $p \parallel q \Rightarrow p' \parallel q'$ can only be originated by *internal* moves.

Now, since $p \Rightarrow p'$, by Proposition 5.13 we have $p' \overset{??}{\Longrightarrow}$; furthermore, since (again by Definition 5.15) $\nexists \mathsf{a} \,.\, p \overset{?\mathsf{a}}{\Longrightarrow} \wedge q \overset{!\mathsf{a}}{\Longrightarrow}$, considering that $p'{\Downarrow^?} \subseteq p{\Downarrow^?}$ and $q'{\Downarrow^!} \subseteq q{\Downarrow^!}$ (because

$q \Rightarrow q'$), we obtain $\nexists \mathsf{a} \,.\, p' \overset{?\mathsf{a}}{\Longrightarrow} \wedge\ q' \overset{!\mathsf{a}}{\Longrightarrow}$. Therefore, by Definition 5.15, we conclude that $p' \parallel q'$ is an unspecified reception configuration for p'. $\square$

As for deadlocks and orphan messages, unspecified reception configurations can be syntactically characterised in the setting of asynchronous session types: intuitively, by Lemma 5.18, unspecified receptions arise when an external choice is paired either with outputs (originated by an internal choice) that it cannot handle, or with another (possibly empty) external choice.

Lemma 5.18. *$T[\sigma] \parallel U[\rho]$ is an unspecified reception configuration for $T[\sigma]$ iff $T \equiv \&_{i\in I} ?\mathsf{a}_i.T_i$ with $I \neq \emptyset$, $\sigma = \epsilon$ and one of the following holds:*

a. *$\rho = !\mathsf{b}.\rho'$, for some ρ' and b such that $\forall i \in I \,.\, \mathsf{a}_i \neq \mathsf{b}$;*

b. *$\rho = \epsilon$ and $U \equiv \bigoplus_{j\in J} !\mathsf{b}_j.U_j$, where $\forall i \in I, j \in J \,.\, \mathsf{a}_i \neq \mathsf{b}_j$;*

c. *$\rho = \epsilon$ and $U \equiv \&_{j\in J} ?\mathsf{b}_j.U_j$.*

Proof. The $\Longrightarrow$ direction follows from Proposition 5.14 and Definition 2.16, considering that, by Definition 5.15:

(i) since $T[\sigma] \overset{??}{\Longrightarrow}$, by Proposition 5.14 T can only be equivalent to a non-empty external choice, and $\sigma = \epsilon$;

(ii) $U[\rho]$ does *not* offer any output matching T's inputs: hence, either U is equivalent to a (possibly empty) external choice and $\rho = \epsilon$ (case *c.* in the statement), or ρ's head exposes an unmatching output — either immediately (case *a.*) or after a τ-move (case *b.*).

The $\Longleftarrow$ direction follows from Definition 2.16: in all cases *a.*–*c.* of the statement, we conclude that $T[\sigma] \parallel U[\rho]$ is an unspecified reception configuration. $\square$

We conclude this section by showing that two behaviours forming an unspecified reception configuration are not I/O compliant.

Lemma 5.19. *If $p \parallel q$ is an unspecified reception configuration for p, then $p \,\ddot{\not\dashv}\, q$.*

Proof. By Definition 5.15 we have $p \overset{??}{\Longrightarrow}$ and $\nexists \mathsf{a} \,.\, p \overset{?\mathsf{a}}{\Longrightarrow} \wedge q \overset{!\mathsf{a}}{\Longrightarrow}$. Then, by Definition 5.11 we have $p{\Downarrow}^{!} = \emptyset$ and $p{\Downarrow}^{?} \neq \emptyset$. Furthermore, $q{\Downarrow}^{!} \cap \mathrm{co}(p{\Downarrow}^{?}) = \emptyset$: this violates clause *a.* of Definition 4.4. Therefore, we have $p \,\ddot{\ntriangleleft}\, q$. □

From Lemma 5.19 we can also easily obtain that, if $p \parallel q$ is an unspecified reception configuration for q, then $p \,\ddot{\ntriangleright}\, q$.

5.4 Safety

We can now define the *safety* of a parallel composition of behaviours: as one might expect from standard definitions (e.g. [LTY15; DY13]), it is the absence of deadlocks, orphan messages and unspecified reception configurations.

Definition 5.20 (Safety). *We say that* $p \parallel q$ is safe *iff* $p \parallel q \Rightarrow p' \parallel q'$ *implies that* $p' \parallel q'$ *is* not *an orphan message configuration,* nor *an unspecified reception configuration.*

Note that Definition 5.20 does *not* explicitly mention absence of deadlocks: this is because, under our definitions, such a property is already implied by the absence of orphan messages and unspecified reception configurations, as shown in Proposition 5.21 (recalling that, by Proposition 5.3, $\dashv\vdash$ coincides with deadlock freedom).

Proposition 5.21. *If $p \parallel q$ is safe, then $p \dashv\vdash q$.*

Proof. We prove the contrapositive. Assume that $p \not\dashv\vdash q$. By Proposition 5.3, there exist p' and q' such that $p' \parallel q'$ is a deadlock state, which, by Definition 5.1, means that:

$$p' \parallel q' \rightarrow \qquad \text{and} \qquad p' \parallel q' \overset{\tau}{\not\rightarrow} \tag{5.1}$$

Note that, by the rightmost part of (5.1), it follows that:

$$p' \overset{\tau}{\not\rightarrow} \qquad \text{and} \qquad q' \overset{\tau}{\not\rightarrow} \tag{5.2}$$

Since, by the leftmost part of (5.1), $p' \parallel q' \rightarrow$, at least one of the behaviours p', q' must take a move, which by (5.2) cannot be labelled τ. Without loss of generality, assume that p' moves. There are two possible (not mutually exclusive) cases:

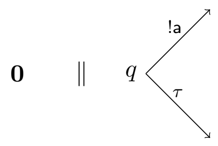


Figure 5.1: The behaviour $\mathbf{0} \parallel q$ is safe, but $\mathbf{0}$ and q are not I/O compliant.

- $p' \xrightarrow{!\mathsf{a}}$. Since $p' \not\xrightarrow{\tau}$, then by Definition 2.6 it must be $p \overset{!!\mathsf{a}}{\Longrightarrow}$. Now, by (5.1) it follows that $q' \not\xrightarrow{?\mathsf{a}}$, otherwise p' and q' would synchronise. Since $q' \not\xrightarrow{\tau}$, this implies that $q' \not\overset{?\mathsf{a}}{\Longrightarrow}$. By Definition 5.5, we obtain that $p' \parallel q'$ is an orphan message configuration, hence by Definition 5.20 we conclude that $p \parallel q$ is not safe.

- $p' \xrightarrow{?\mathsf{a}}$. Since $p' \not\xrightarrow{\tau}$, then by Definition 5.11 it must be $p' \overset{??}{\Longrightarrow}$. Now, by (5.1) it follows that $\nexists \mathsf{b} : p' \overset{?\mathsf{b}}{\Longrightarrow} \wedge q' \overset{!\mathsf{b}}{\Longrightarrow}$, otherwise p' and q' would synchronise. By Definition 5.15, we obtain that $p' \parallel q'$ is an unspecified reception configuration, hence by Definition 5.20 we conclude that $p \parallel q$ is not safe. □

In Theorem 5.22, we prove that I/O compliance implies safe interactions: this formalises the "correctness" claim we made in Chapter 4.

Theorem 5.22. *If $p \mathrel{\dot{\bowtie}} q$, then $p \parallel q$ is safe.*

Proof. Let $p \parallel q \Rightarrow p' \parallel q'$. By Proposition B.1 and Proposition B.2, we have $p' \mathrel{\dot{\bowtie}} q'$. Therefore, by Lemma 5.4 we have that $p' \parallel q'$ is *not* a deadlock state. Moreover, by Lemma 4.5 and Lemma 5.9, we have that $p' \parallel q'$ is *not* an orphan message configuration. Finally, by Lemma 4.5 and Lemma 5.19, we have that $p' \parallel q'$ is *not* an unspecified reception configuration. Hence, by Definition 5.20 we conclude that $p \parallel q$ is safe. □

The following example shows that the converse of Theorem 5.22 does not hold.

Example 5.23. *The behaviour $\mathbf{0} \parallel q$ in Figure 5.1 is safe. Indeed, we have that:*

- *neither $\mathbf{0}$ nor q have persistent outputs, and so by Definition 5.5 it follows that $\mathbf{0} \parallel q$ has no orphan messages;*

- *neither $\mathbf{0}$ nor q is an input state, and so by Definition 5.15 it follows that $\mathbf{0} \parallel q$ is not an unspecified reception configuration.*

Note that $\mathbf{0} \not\mathrel{\dot{\bowtie}} q$: indeed, $!\mathsf{a} \in q{\Downarrow}^{!} \not\subseteq \mathrm{co}(\mathbf{0}{\Downarrow}^{?}) = \emptyset$, and therefore item a. of Definition 4.4 is false.

Albeit safety does *not* imply I/O compliance on general behaviours, Theorem 5.24 below shows that this implication actually holds when considering asynchronous session types.

Theorem 5.24. *If $T[\sigma] \parallel U[\rho]$ is safe, then $T[\sigma] \mathrel{\ddot{\bowtie}} U[\rho]$.*

Proof. We prove the contrapositive. Assume that $T[\sigma] \mathrel{\not\ddot{\bowtie}} U[\rho]$. By Definition 4.4, this implies that there exist T', U', σ', ρ' such that $T[\sigma] \parallel U[\rho] \Rightarrow T'[\sigma'] \parallel U'[\rho']$ and item *a.* is false for $p = T'[\sigma']$ and $q = U'[\rho']$. There are the following two cases:

1. $p{\Downarrow^!} \not\subseteq \mathrm{co}(q{\Downarrow^?})$. Then, there exists some a such that $!\mathsf{a} \in p{\Downarrow^!}$ and $?\mathsf{a} \notin q{\Downarrow^?}$. Since $!\mathsf{a} \in p{\Downarrow^!}$, one of the following two cases must hold:

(i) $\sigma' = !\mathsf{a}.\sigma''$. By Proposition 2.22 it follows that $T'[\sigma'] \xRightarrow{!!\mathsf{a}}$. Since $?\mathsf{a} \notin q{\Downarrow^?}$, we have that $U'[\rho'] \overset{?\mathsf{a}}{\not\Rightarrow}$. Therefore, by Lemma 5.7 we obtain that $T'[\sigma'] \parallel U'[\rho']$ is an orphan message configuration. By Definition 5.20, we conclude that $T[\sigma] \parallel U[\rho]$ is not safe.

(ii) $\sigma' = \epsilon$ and $T' \equiv !\mathsf{a}.T'' \oplus T'''$. By Definition 2.16 we have the transition:

$$T'[\epsilon] \parallel U'[\rho'] \xrightarrow{\tau} T''[!\mathsf{a}] \parallel U'[\rho']$$

and we fall back to the previous case.

2. $\big((p{\Downarrow^!} = \emptyset \land p{\Downarrow^?} \neq \emptyset) \not\Longrightarrow \emptyset \neq q{\Downarrow^!} \subseteq \mathrm{co}(p{\Downarrow^?})\big)$. Then, $p{\Downarrow^!} = \emptyset \land p{\Downarrow^?} \neq \emptyset$. This implies that T' is equivalent to an external choice, and that $\sigma' = \epsilon$. Furthermore, one of the following two cases must hold:

- $q{\Downarrow^!} = \emptyset$. This implies that $\rho' = \epsilon$, and U' is equivalent to an external choice (possibly empty). By item *c.* of Lemma 5.18, it follows that $T'[\sigma'] \parallel U'[\rho']$ is an unspecified reception configuration. By Definition 5.20, we conclude that $T[\sigma] \parallel U[\rho]$ is not safe.
- there exists some a such that $!\mathsf{a} \in q{\Downarrow^!}$ and $?\mathsf{a} \notin p{\Downarrow^?}$. Therefore, either one of the following subcases applies:
  - $\rho' = !\mathsf{a}.\rho''$. Then, by item *a.* of Lemma 5.18, it follows that $T'[\sigma'] \parallel U'[\rho']$ is an unspecified reception configuration. By Definition 5.20 we conclude that $T[\sigma] \parallel U[\rho]$ is not safe.

– $\rho' = \epsilon$ and $U' \equiv \mathsf{!a}.U'' \oplus U'''$. By Definition 2.16 we have the transition:

$$T'[\epsilon] \parallel U'[\epsilon] \xrightarrow{\tau} T'[\epsilon] \parallel U''[\mathsf{!a}]$$

and we fall back to the previous case. $\square$

Therefore, by combining Theorem 5.22 and Theorem 5.24, we obtain that safety and I/O compliance coincide in the setting of asynchronous session types.

Chapter 6

I/O simulation

In this chapter we address the topics of Section 1.1.1 by introducing $\ddot{\leqslant}$, a simulation relation between general behaviours, that generalises the usual notions of session typing and subtyping, with the typical co/contra-variance of outputs and inputs.

6.1 Introducing I/O simulation

We start by adapting to our framework one of the classical notions of *sub-behaviour* from the literature on behavioural contracts: the *strong subcontract relation* of [CGP09]. A behaviour p is a subcontract of p' iff, whenever p' is compliant with some (arbitrary) behaviour q, then p is compliant with q[1]. Thus, p can transparently replace p', in all contexts.

Definition 6.1 (Subcontract relation)**.** *We define the relation $\sqsubseteq$ between behaviours as:*

$$p \sqsubseteq q \qquad \textit{iff} \qquad \forall r \in \mathbb{U} \,.\, q \mathrel{\ddot{\bowtie}} r \ \ \textit{implies} \ \ p \mathrel{\ddot{\bowtie}} r$$

We write $p \sqsubseteq_{\mathbb{R}} q$ to restrict r to the set of behaviours $\mathbb{R}$ (i.e., $\forall r \in \mathbb{R} \ldots$).

The main difference between Definition 6.1 and the subcontract relation in [CGP09] lies in the underlying notion of "correct" interaction: we require (symmetric) I/O compliance in each context, while [CGP09] only requires progress.

[1]In this work, the direction of $\sqsubseteq$ is opposite w.r.t. the subcontract relation in [CGP09]: this topic will be further discussed in Section 9.4.

Despite its elegance and generality, Definition 6.1 cannot be directly exploited to establish whether two behaviours are related, due to the universal quantification over all contexts. For session types, alternative characterisations of $\sqsubseteq$ have been defined, usually in the form of a syntax-driven coinductive relation [CGP09; BL10]. This approach amounts to restricting p, q and r in Definition 6.1 to a process calculus with specific syntax and transition rules — e.g., $p, q, r \in \mathbb{U}_{\mathrm{ST}}$. In our semantic framework, however, behaviours are not syntax. We shall extend these characterisations from session behaviours to arbitrary ones, without resorting to a universal quantification over contexts. To do that, we define an *I/O simulation* relation on behaviours, denoted by $\ddot{\leqslant}$. We show that it is a preorder (Theorem 6.12), and it preserves I/O compliance (Theorem 6.13). $\ddot{\leqslant}$ is equivalent to the subtype relation on sync session behaviours (Theorem 6.16), albeit stricter on arbitrary behaviours. Notice that our definition exploits the transition relation $\Rrightarrow$ introduced in Definition 2.7.

Definition 6.2 (I/O simulation)**.** $\ddot{\mathcal{R}}$ *is an* I/O simulation relation *iff, whenever* $p \;\ddot{\mathcal{R}}\; q$*, then* $\exists \mathbb{Q}$ *(called* predictive set*) such that* $q \Rrightarrow \mathbb{Q}$*, and:*

a. $p{\Downarrow}^{!} = \emptyset \implies \mathbb{Q}{\Downarrow}^{!} = \emptyset$;

b. $\mathbb{Q}{\Downarrow}^{??} \subseteq p{\Downarrow}^{?} \;\wedge\; (\mathbb{Q}{\Downarrow}^{?} = \emptyset \implies p{\Downarrow}^{?} = \emptyset)$;

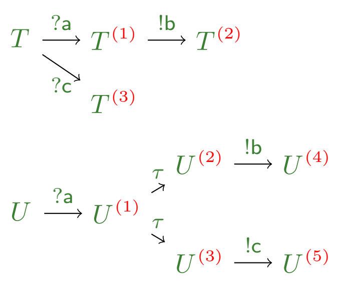
c. $p \xrightarrow{\tau} p' \implies \exists q' \,.\, \mathbb{Q} \Rightarrow q' \wedge p' \;\ddot{\mathcal{R}}\; q'$;

d. $p \xrightarrow{!\mathsf{a}} p' \implies \exists q' \,.\, \mathbb{Q} \xRightarrow{!\mathsf{a}} q' \wedge p' \;\ddot{\mathcal{R}}\; q'$;

e. $p \xrightarrow{?\mathsf{a}} p' \wedge \mathbb{Q} \xRightarrow{??\mathsf{a}} \implies \exists q' \,.\, \mathbb{Q} \xRightarrow{?\mathsf{a}} q' \wedge p' \;\ddot{\mathcal{R}}\; q'$.

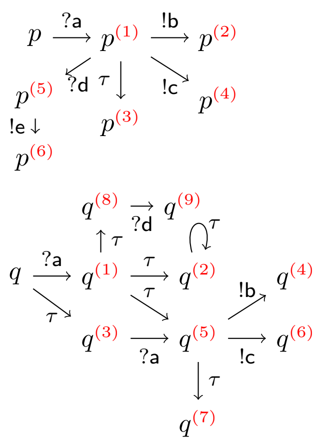
We write $\ddot{\leqslant}$ *for the largest I/O simulation,* $\ddot{\approx}$ *for the largest symmetric I/O simulation, and* $\ddot{=}$ *for* $\ddot{\leqslant} \cap \ddot{\geqslant}$.

Definition 6.2 can be explained in terms of a sort of simulation game between players p and q. At the first step, q predicts a suitable choice of its internal moves, via a set $\mathbb{Q}$ of states reachable from q. The outputs of $\mathbb{Q}$ must include those of p (item *d.*), and the *weakly persistent* inputs of $\mathbb{Q}$ (Definition 2.6) must be included in the inputs of p (item *b.*). Moreover, if p has no outputs, then also $\mathbb{Q}$ cannot have outputs, and if $\mathbb{Q}$ has no inputs, then also p cannot have inputs (items *a.*,*b.*). Intuitively, these constraints reflect subtyping in session types: inputs (external choices) can be enlarged (if not empty),



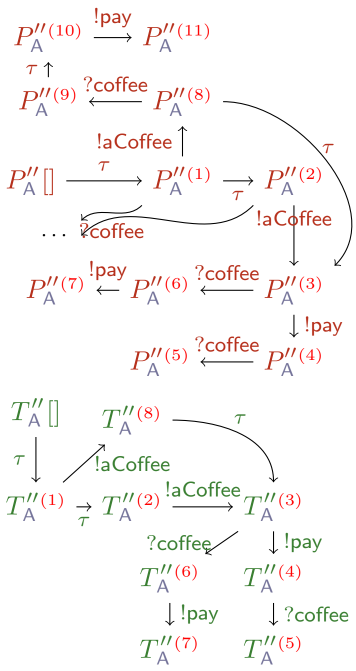
Relation	Pred. set
(T, U)	$\{U\}$
$(T^{(1)}, U^{(1)})$	$\{U^{(1)}\}$
$(T^{(1)}, U^{(2)})$	$\{U^{(2)}\}$
$(T^{(2)}, U^{(4)})$	$\{U^{(4)}\}$

$T^{(1)}$ is in relation with $U^{(1)}, U^{(2)}$: both the latter match the outputs of the first. $T^{(3)}$ is *not* in the relation: it is reached via an input ?c unmatched by U. $U^{(3)}, U^{(5)}$ are *not* in the relation: they are reached via a τ and an output !c unmatched by T.



Relation	Pred. set
(p, q)	$\{q\}$
$(p, q^{(3)})$	$\{q^{(3)}\}$
$(p^{(1)}, q^{(1)})$	$\{q^{(1)}\}$
$(p^{(1)}, q^{(5)})$	$\{q^{(5)}\}$
$(p^{(2)}, q^{(4)})$	$\{q^{(4)}\}$
$(p^{(3)}, q^{(2)})$	$\{q^{(2)}\}$
$(p^{(3)}, q^{(7)})$	$\{q^{(7)}\}$
$(p^{(4)}, q^{(6)})$	$\{q^{(6)}\}$

p is in relation with $q, q^{(3)}$, matching their ?a. Then, $p^{(1)}$ can either perform !b, !c, ?d or quit: this is matched by $q^{(1)}$, $q^{(5)}$; if $p^{(1)}$ follows its τ-branch to $p^{(3)}$, the latter is related with $q^{(2)}$, $q^{(7)}$. Note that $q^{(2)}$ does not stop, but performs τ-loop. Also note that the relation does *not* include $p^{(5)}$ since $q^{(1)} \overset{?\mathsf{d}}{\Longrightarrow}$ but $\{q^{(1)}\} \overset{??\mathsf{d}}{\not\Longrightarrow}$; indeed, it *cannot* include $p^{(5)}$, due to its !e (unmatched in q's reductions).



Relation	Pred. set
$(P''_{\mathsf{A}}[], T''_{\mathsf{A}}[])$	$\{T''^{(1)}_{\mathsf{A}}\}$
$(P''^{(1)}_{\mathsf{A}}, T''^{(1)}_{\mathsf{A}})$	$\{T''^{(1)}_{\mathsf{A}}\}$
$(P''^{(2)}_{\mathsf{A}}, T''^{(2)}_{\mathsf{A}})$	$\{T''^{(2)}_{\mathsf{A}}\}$
$(P''^{(3)}_{\mathsf{A}}, T''^{(3)}_{\mathsf{A}})$	$\{T''^{(3)}_{\mathsf{A}}\}$
$(P''^{(4)}_{\mathsf{A}}, T''^{(4)}_{\mathsf{A}})$	$\{T''^{(4)}_{\mathsf{A}}\}$
$(P''^{(5)}_{\mathsf{A}}, T''^{(5)}_{\mathsf{A}})$	$\{T''^{(5)}_{\mathsf{A}}\}$
$(P''^{(6)}_{\mathsf{A}}, T''^{(6)}_{\mathsf{A}})$	$\{T''^{(6)}_{\mathsf{A}}\}$
$(P''^{(7)}_{\mathsf{A}}, T''^{(7)}_{\mathsf{A}})$	$\{T''^{(7)}_{\mathsf{A}}\}$
$(P''^{(8)}_{\mathsf{A}}, T''^{(3)}_{\mathsf{A}})$	$\{T''^{(3)}_{\mathsf{A}}\}$
$(P''^{(9)}_{\mathsf{A}}, T''^{(6)}_{\mathsf{A}})$	$\{T''^{(6)}_{\mathsf{A}}\}$
$(P''^{(10)}_{\mathsf{A}}, T''^{(6)}_{\mathsf{A}})$	$\{T''^{(6)}_{\mathsf{A}}\}$
$(P''^{(11)}_{\mathsf{A}}, T''^{(7)}_{\mathsf{A}})$	$\{T''^{(7)}_{\mathsf{A}}\}$

Note that $P''^{(1)}_{\mathsf{A}}$ and $P''^{(2)}_{\mathsf{A}}$ have an input branch (?coffee) unmatched by $T''^{(1)}_{\mathsf{A}}$ and $T''^{(2)}_{\mathsf{A}}$, and thus leading to states not considered in the relation (and here omitted).

$$
\begin{aligned}
T''_{\mathsf{A}}[] &= \mathsf{!aCoffee.!pay.?coffee}\ []\\
T''^{(1)}_{\mathsf{A}} &= \mathsf{!pay.?coffee}\ [\mathsf{!aCoffee}]\\
T''^{(2)}_{\mathsf{A}} &= \mathsf{?coffee}\ [\mathsf{!aCoffee.!pay}]\\
T''^{(3)}_{\mathsf{A}} &= \mathsf{?coffee}\ [\mathsf{!pay}]\\
T''^{(4)}_{\mathsf{A}} &= \mathsf{?coffee}\ []\\
T''^{(5)}_{\mathsf{A}} &= \mathbf{0}\ [] \ = \ T''^{(7)}_{\mathsf{A}}\\
T''^{(6)}_{\mathsf{A}} &= \mathbf{0}\ [\mathsf{!pay}]\\
T''^{(7)}_{\mathsf{A}} &= \mathbf{0}\ [] \ = \ T''^{(5)}_{\mathsf{A}}\\
T''^{(8)}_{\mathsf{A}} &= \mathsf{!pay.?coffee}\ []
\end{aligned}
$$

Table 6.1: Examples of I/O simulation. For each pair of behaviours on the left, the table shows an I/O simulation relation, and the predictive sets supporting each pair of related states.

while outputs (internal choices) can be narrowed (but not emptied). The requirements above must be preserved by the moves of p: τ-moves and outputs of p must be (weakly) simulated by some process in $\mathbb{Q}$ (items *c.*,*d.*); the same holds for p's inputs (item *e.*), but only moves shared by p and $\mathbb{Q}$, and persistent in the latter, are considered.

$$\begin{tikzcd}
p \arrow[r, "?\mathsf{a}"] \arrow[dr, "?\mathsf{b}"'] & p^{(1)} \arrow[r, "!\mathsf{x}"] & p^{(2)} \\
 & p^{(3)} \arrow[r, "!\mathsf{y}"'] & p^{(4)}
\end{tikzcd}$$

$$\begin{tikzcd}
 & q^{(5)} \arrow[r, "!\mathsf{c}"] & q^{(6)} & \\
q \arrow[ur, "\tau"] \arrow[r, "\tau"] \arrow[ddr, "\tau"] & q^{(01)} \arrow[r, "?\mathsf{a}"] \arrow[dr, "?\mathsf{b}"'] & q^{(1)} \arrow[r, "!\mathsf{x}"] & q^{(2)} \\
 & & q^{(31)} & \\
 & q^{(02)} \arrow[r, "?\mathsf{b}"'] \arrow[ur, "?\mathsf{a}"] & q^{(3)} \arrow[r, "!\mathsf{y}"'] & q^{(4)}
\end{tikzcd}$$

Relation	Pred. set
(p, q)	$\{q^{(01)}, q^{(02)}\}$
$(p^{(1)}, q^{(1)})$	$\{q^{(1)}\}$
$(p^{(2)}, q^{(2)})$	$\{q^{(2)}\}$
$(p^{(3)}, q^{(3)})$	$\{q^{(3)}\}$
$(p^{(4)}, q^{(4)})$	$\{q^{(4)}\}$

p is related with q, with a predictive set containing 2 elements. This is the only possible predictive set supporting the relation: in fact, if the predictive set included q or $q^{(5)}$, then p would be forced to offer some output (by clause *a.* of Definition 6.2); moreover, if the predictive set did *not* contain $q^{(01)}$, then the ?a-branch of p would need to match the ?a-branch of $q^{(02)}$ (by clause *e.* of Definition 6.2) — but in this case we would need $(p^{(1)}, q^{(31)}) \in \ddot{\lessgtr}$, which is false. A similar problem would arise if the predictive set did not contain $q^{(02)}$.

Table 6.2: Another example of I/O simulation.

Example 6.3. *In Table 6.1 we exemplify $\ddot{\lessgtr}$. The first pair of behaviours correspond to the sync session types $T = ?\mathsf{a}.!\mathsf{b} \,\&\, ?\mathsf{c}$ and $U = ?\mathsf{a}.(!\mathsf{b} \oplus !\mathsf{c})$, and show how $\ddot{\lessgtr}$ is covariant w.r.t. outputs, and contravariant w.r.t. inputs. The third pair corresponds to Alice's asynchronous type T''_{A} and her process P''_{A}, from Chapter 3: we show that the former I/O simulates the latter.*

Example 6.4. *Consider Figure 6.1. To assess $p \ddot{\lessgtr} q$, we choose a predictive set $\mathbb{Q}$ that mandates the inputs of p, and includes its outputs (note that p has an additional input* ?c′ not *offered by $\mathbb{Q}$). The same happens with the predictive set $\mathbb{R}$, assessing $q \ddot{\lessgtr} r$. Note that $\mathbb{R}$ and the small set inside are also predictive sets for $p \ddot{\lessgtr} r$.*

Example 6.5. *Table 6.2 shows another example of I/O simulation, where the predictive set for the pair $(p, q) \in \ddot{\lessgtr}$ contains more than one element.*

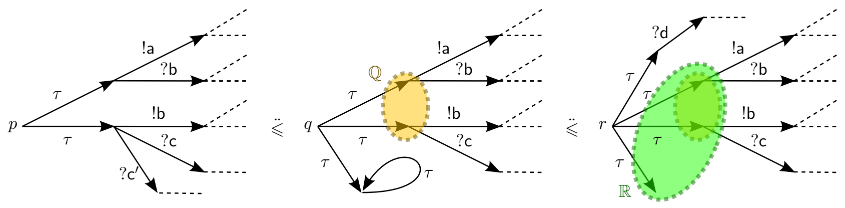


Figure 6.1: I/O simulation. $\mathbb{Q}$, $\mathbb{R}$ are the predictive sets resp. for $p \ddot{\lessgtr} q$ and $q \ddot{\lessgtr} r$.

6.2 Basic properties

In this section, we introduce some properties of the subcontract relation $\sqsubseteq$ and I/O simulation $\ddot{\leqslant}$.

The following lemma gives a syntactic characterisation of pairs of synchronous session behaviours in the subcontract relation.

Lemma 6.6 ($\sqsubseteq_{\mathbb{U}_{\text{ST}}}$-induced shapes of session types)**.** *$T \sqsubseteq_{\mathbb{U}_{ST}} U$ implies either (up-to unfolding):*

a. $T = U = \mathbf{0}$*;*

b. $T = \&_{k \in K} ?\mathsf{a}_k . T_k$ *and* $U = \&_{i \in I} ?\mathsf{a}_i . T_i$*, with* $\emptyset \neq I \subseteq K$ *and* $\forall i \in I \,.\, T_i \sqsubseteq_{\mathbb{U}_{ST}} U_i$*;*

c. $T = \bigoplus_{k \in K} !\mathsf{a}_k . T_k$ *and* $U = \bigoplus_{i \in I} !\mathsf{a}_i . T_i$*, with* $\emptyset \neq K \subseteq I$*, and* $\forall k \in K . T_k \sqsubseteq_{\mathbb{U}_{ST}} U_k$*.*

Proof. See page 125. □

The following lemma shows that, to determine if two synchronous session behaviours are in the subcontract relation (i.e., $T \sqsubseteq U$), we do *not* need to consider all possible behaviours $r \in \mathbb{U}$ such that $U \ddot{\bowtie} r$: indeed, we can restrict to the case where r is a synchronous session behaviour itself.

Lemma 6.7. $T \sqsubseteq U \iff T \sqsubseteq_{\mathbb{U}_{ST}} U$*.*

Proof. See page 132. □

We now study some properties of $\ddot{\leqslant}$. Lemma 6.8 ensures that Definition 6.2 is well-formed.

Lemma 6.8. *Let $\ddot{\mathbb{R}}$ be a set of I/O simulations. Then, $\bigcup \ddot{\mathbb{R}}$ is an I/O simulation.*

Proof. See page 126. □

The following result relates I/O simulation with weak moves. When $p \ddot{\leqslant} q$, the relation $\ddot{\leqslant}$ is preserved by forward τ-moves of p and backward τ-moves of q.

Lemma 6.9. *If $p \ddot{\leqslant} q$, with $p \Rightarrow p'$ and $q' \Rightarrow q$, then $p' \ddot{\leqslant} q'$.*

Proof. See page 128. □

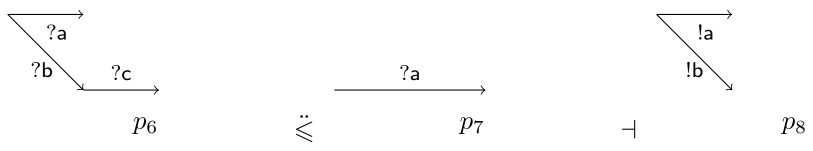


Figure 6.2: Progress is not preserved by I/O simulation (on general behaviours).

A consequence of Lemma 6.9 is that I/O simulation reflects the internal non-determinism of a behaviour: each τ-move gives a reduct which is “smaller” than its redex, as formalised in Corollary 6.10.

Corollary 6.10 ($\ddot{\leqslant}$ reflects internal non-determinism)**.** *If $p \Rightarrow p'$, then $p' \ddot{\leqslant} p$.*

Proof. From Lemma C.1 we have $p \ddot{\leqslant} p$, and by Lemma C.2 we conclude $p' \ddot{\leqslant} p$. □

Weak simulation ($\precapprox$) and I/O simulations are unrelated, i.e. $\ddot{\leqslant} \not\subseteq \precapprox \not\subseteq \ddot{\leqslant}$ (see Proposition C.12). However, weak *bisimulation* ($\approx$) is strictly stronger than I/O bisimulation.

Theorem 6.11. $\approx \subsetneq \ddot{\approx}$

Proof. See page 132. □

By Theorem 6.12, $\ddot{\leqslant}$ is a preorder, as it is the case for the subtype relation. This is not quite straightforward, due to the existential quantification on the predictive set $\mathbb{Q}$.

Theorem 6.12. $(\mathbb{U}, \ddot{\leqslant})$ *is a preorder.*

Proof. See page 129. □

6.3 On I/O simulation and I/O compliance

Quite surprisingly, on general behaviours *progress* is *not* preserved by $\ddot{\leqslant}$: if $p \ddot{\leqslant} q \dashv r$, then it is not always the case that $p \dashv r$. For instance, consider the behaviours in Figure 6.2. It is easy to check that $p_6 \ddot{\leqslant} p_7$ and $p_7 \dashv p_8$. However, $p_6 \not\dashv p_8$: indeed, if p_8 chooses the branch !b, then p_6 is stuck waiting for ?c.

Theorem 6.13 is one of our main results: it states that $\ddot{\leqslant}$ preserves symmetric *I/O compliance*. This is a further motivation for using $\dot{\bowtie}$ instead of $\dashv\vdash$, when dealing with behaviours where these two notions do not coincide. In the example above, p_8 is not a sync session behaviour: were all behaviours in Figure 6.2 elements of $\mathbb{U}_{\mathrm{ST}}$, we would also have preserved progress (by Theorem 4.9).

Theorem 6.13. *$p \mathrel{\ddot{\leqslant}} q \circ r \implies p \circ r$, for $\circ \in \{\ddot{\rhd}, \ddot{\bowtie}\}$.*

Proof. We first prove the statement for $\circ = \ddot{\rhd}$. Let:

$$\mathcal{R} \;=\; \big\{(p,r) \,\big|\, \exists q \;.\; p \mathrel{\ddot{\leqslant}} q \wedge q \mathrel{\ddot{\rhd}} r\big\}$$

We will show that $\mathcal{R}$ is an I/O compliance relation between p and r (in the inverse order). Let $(p,r) \in \mathcal{R}$, via some q such that $p \mathrel{\ddot{\leqslant}} q$ and $r \mathrel{\ddot{\lhd}} q$. Now, let $\mathbb{Q}$ be the predictive set supporting the pair $(p,q) \in \ddot{\leqslant}$, and let us examine the clauses of Definition 4.4.

item *a.* From $p \mathrel{\ddot{\leqslant}} q$ (item *b.* of Definition 6.2) we have $\mathbb{Q}{\Downarrow}^{??} \subseteq p{\Downarrow}^{?}$; furthermore, from $r \mathrel{\ddot{\lhd}} q$ (by Proposition B.3) we have $r{\Downarrow}^{!} \subseteq \mathrm{co}(q{\Downarrow}^{??})$, and from Proposition C.6 we obtain:

$$r{\Downarrow}^{!} \subseteq \mathrm{co}(\mathbb{Q}{\Downarrow}^{??}) \tag{6.1}$$

Furthermore, since $\forall q' \in \mathbb{Q} \,.\, q \Rightarrow q'$, from $r \mathrel{\ddot{\lhd}} q$ and Proposition B.1 we have $\forall q' \in \mathbb{Q} \,.\, r \mathrel{\ddot{\lhd}} q'$. Hence, by item *b.* of Definition 6.2:

$$\forall q' \in \mathbb{Q} \;.\; \Big(r{\Downarrow}^{!} = \emptyset \wedge r{\Downarrow}^{?} \neq \emptyset \implies \emptyset \neq q'{\Downarrow}^{!} \subseteq \mathrm{co}(r{\Downarrow}^{?})\Big)$$

and therefore:

$$r{\Downarrow}^{!} = \emptyset \wedge r{\Downarrow}^{?} \neq \emptyset \implies \emptyset \neq \mathbb{Q}{\Downarrow}^{!} \subseteq \mathrm{co}(r{\Downarrow}^{?}) \tag{6.2}$$

From $p \mathrel{\ddot{\leqslant}} q$, by Definition 6.2 we have:

- $p{\Downarrow}^{!} = \emptyset \implies \mathbb{Q}{\Downarrow}^{!} = \emptyset$ (item *a.* of Definition 6.2), and therefore $\mathbb{Q}{\Downarrow}^{!} \neq \emptyset \implies p{\Downarrow}^{!} \neq \emptyset$;
- $p{\Downarrow}^{!} \subseteq \mathbb{Q}{\Downarrow}^{!}$ (item *d.*)
- $\mathbb{Q}{\Downarrow}^{??} \subseteq p{\Downarrow}^{?}$ (item *b.*).

Summing up:

$$\mathbb{Q}{\Downarrow}^{??} \subseteq p{\Downarrow}^{?} \quad \text{and} \quad \mathbb{Q}{\Downarrow}^{!} \neq \emptyset \implies \emptyset \neq p{\Downarrow}^{!} \subseteq \mathbb{Q}{\Downarrow}^{!} \tag{6.3}$$

and combining Equation (6.3) with Equations (6.1) and (6.2), we conclude:

$$\Big(r{\Downarrow}^{!} \subseteq \mathrm{co}(p{\Downarrow}^{?})\Big) \;\wedge\; \Big(r{\Downarrow}^{!} = \emptyset \wedge r{\Downarrow}^{?} \neq \emptyset \implies \emptyset \neq p{\Downarrow}^{!} \subseteq \mathrm{co}(r{\Downarrow}^{?})\Big)$$

item *b.* We have to show that $p \xrightarrow{\ell} p'$ and $r \xrightarrow{\mathrm{co}(\ell)} r'$ implies $(p', r') \in \mathcal{R}$. Assuming the premise, we have two cases, depending on whether ℓ is an input or an output action:

- $\ell = \mathsf{!a}$. Then, by item *d.* of Definition 6.2, $\exists q' \,.\, \mathbb{Q} \overset{\mathsf{!a}}{\Rightarrow} q' \wedge p' \mathrel{\ddot{\leqslant}} q'$. Now, from $q \mathrel{\ddot{\rhd}} r$ and Proposition B.2 we have $q' \mathrel{\ddot{\rhd}} r'$: we conclude $(p', r') \in \mathcal{R}$;
- $\ell = \mathsf{?a}$. Then, $r \xrightarrow{\mathsf{!a}} r'$; hence, from $q \mathrel{\ddot{\rhd}} r$ and Proposition B.3 we have $q \overset{\mathsf{??a}}{\Longrightarrow}$, and by Proposition C.6 we obtain $\mathbb{Q} \overset{\mathsf{??a}}{\Longrightarrow}$. Thus, by item *e.* of Definition 6.2, $\exists q' \,.\, \mathbb{Q} \overset{\mathsf{?a}}{\Rightarrow} q' \wedge p' \mathrel{\ddot{\leqslant}} q'$. Now, from $q \mathrel{\ddot{\rhd}} r$ and Proposition B.2 we have $q' \mathrel{\ddot{\rhd}} r'$: we conclude $(p', r') \in \mathcal{R}$;

item *c.* From $p \mathrel{\ddot{\leqslant}} q$ and Lemma C.2, we have that $p \xrightarrow{\tau} p'$ implies $p' \mathrel{\ddot{\leqslant}} q$: we conclude $(p', r) \in \mathcal{R}$;

item *d.* From $q \mathrel{\ddot{\rhd}} r$, by item *d.* of Definition 4.4, we know that $r \xrightarrow{\tau} r'$ implies $q \mathrel{\ddot{\rhd}} r'$: we conclude $(p, r') \in \mathcal{R}$.

This concludes the proof for $\circ = \mathrel{\ddot{\rhd}}$.

The proof for $\circ = \mathrel{\ddot{\bowtie}}$ follows a similar approach, but this time we let:

$$\mathcal{R} \;=\; \big\{(p, r) \,\big|\, \exists q \;.\; p \mathrel{\ddot{\leqslant}} q \wedge q \mathrel{\ddot{\bowtie}} r\big\} \tag{6.4}$$

Now, we prove that $\mathcal{R}$ is a symmetric I/O compliance relation. Since $\mathrel{\ddot{\bowtie}} = \mathrel{\ddot{\rhd}} \cap \mathrel{\ddot{\lhd}}$ (by Lemma 4.5), most of the proof is already developed above, when $\circ = \mathrel{\ddot{\rhd}}$. The only difference is that, in the *symmetric* version of item *a.* of Definition 4.4, we have the following *additional* clause, introduced by $\mathrel{\ddot{\lhd}}$, that needs to be satisfied:

$$\Big(p{\Downarrow}^{!} \subseteq \mathrm{co}(r{\Downarrow}^{?}) \;\wedge\; \Big(p{\Downarrow}^{!} = \emptyset \wedge p{\Downarrow}^{?} \neq \emptyset \implies \emptyset \neq r{\Downarrow}^{!} \subseteq \mathrm{co}(p{\Downarrow}^{?})\Big)\Big) \tag{6.5}$$

We proceed by proving such a clause, for each $(p, r) \in \mathcal{R}$ and corresponding q from Equation (6.4). From $p \mathrel{\ddot{\leqslant}} q$ (item *d.* of Definition 6.2) we have $p{\Downarrow}^{!} \subseteq \mathbb{Q}{\Downarrow}^{!} \subseteq q{\Downarrow}^{!}$, and from $q \ddot{\lhd} r$ (item *a.* of Definition 4.4) we have $q{\Downarrow}^! \subseteq \mathrm{co}(r{\Downarrow}^?)$; therefore,

$$p{\Downarrow}^! \subseteq \mathrm{co}(r{\Downarrow}^?) \tag{6.6}$$

Furthermore, from $q \ddot{\lhd} r$ (item *a.* of Definition 4.4) we have:

$$q{\Downarrow}^! = \emptyset \wedge q{\Downarrow}^? \neq \emptyset \implies \emptyset \neq r{\Downarrow}^! \subseteq \mathrm{co}(q{\Downarrow}^?) \tag{6.7}$$

If the premise holds, from $q \ddot{\lhd} r$ and Proposition B.5, we have $r{\Downarrow}^! \subseteq \mathrm{co}(q{\Downarrow}^{??})$, and from Proposition C.6 we have $q{\Downarrow}^{??} \subseteq \mathbb{Q}{\Downarrow}^{??}$ — and therefore $r{\Downarrow}^! \subseteq \mathrm{co}(\mathbb{Q}{\Downarrow}^{??})$. Combining these observations with Equation (6.7), we obtain:

$$q{\Downarrow}^! = \emptyset \wedge q{\Downarrow}^? \neq \emptyset \implies \emptyset \neq r{\Downarrow}^! \subseteq \mathrm{co}(\mathbb{Q}{\Downarrow}^{??}) \tag{6.8}$$

Since $\forall q' \in \mathbb{Q} \,.\, q \Rightarrow q'$, by Proposition B.1 we have $\forall q' \in \mathbb{Q} \,.\, q' \ddot{\lhd} r$. Then, from Equation (6.8) we have:

$$\forall q' \in \mathbb{Q} \,.\, q'{\Downarrow}^! = \emptyset \wedge q'{\Downarrow}^? \neq \emptyset \implies \emptyset \neq r{\Downarrow}^! \subseteq \mathrm{co}(\mathbb{Q}{\Downarrow}^{??}) \tag{6.9}$$

and therefore:

$$\mathbb{Q}{\Downarrow}^! = \emptyset \wedge \mathbb{Q}{\Downarrow}^? \neq \emptyset \implies \emptyset \neq r{\Downarrow}^! \subseteq \mathrm{co}(\mathbb{Q}{\Downarrow}^{??}) \tag{6.10}$$

From $p \ddot{\lessgtr} q$, by Definition 6.2 we have:

- $p{\Downarrow}^! = \emptyset \implies \mathbb{Q}{\Downarrow}^! = \emptyset$ (item *a.*);
- $\mathbb{Q}{\Downarrow}^{??} \subseteq p{\Downarrow}^?$ (item *b.*), and therefore $\mathrm{co}(\mathbb{Q}{\Downarrow}^{??}) \subseteq \mathrm{co}(p{\Downarrow}^?)$;
- $\mathbb{Q}{\Downarrow}^? = \emptyset \implies p{\Downarrow}^? = \emptyset$ (item *b.*), and therefore $p{\Downarrow}^? \neq \emptyset \implies \mathbb{Q}{\Downarrow}^? \neq \emptyset$.

Summing up:

$$p{\Downarrow}^! = \emptyset \wedge p{\Downarrow}^? \neq \emptyset \implies \mathbb{Q}{\Downarrow}^! = \emptyset \wedge \mathbb{Q}{\Downarrow}^? \neq \emptyset \wedge \mathrm{co}(\mathbb{Q}{\Downarrow}^{??}) \subseteq \mathrm{co}(p{\Downarrow}^?) \tag{6.11}$$

and combining Equation (6.11) with Equations (6.3) and (6.6), we conclude:

$$\Big(p{\Downarrow}^! \subseteq \mathrm{co}(r{\Downarrow}^?)\Big) \wedge \Big(p{\Downarrow}^! = \emptyset \wedge p{\Downarrow}^? \neq \emptyset \implies \emptyset \neq r{\Downarrow}^! \subseteq \mathrm{co}(p{\Downarrow}^?)\Big)$$

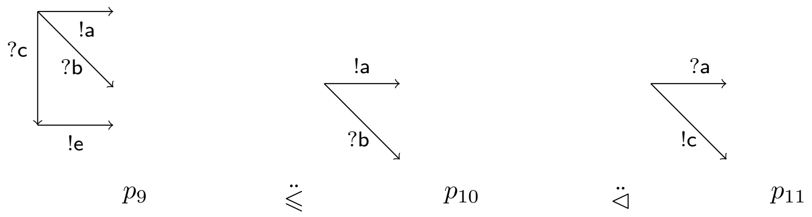


Figure 6.3: $\ddot{\lhd}$ is not preserved by I/O simulation (on general behaviours).

which corresponds to Equation (6.5).

The proofs for the remaining items *b.–d.* can be obtained from the ones provided for $\circ = \ddot{\rhd}$, simply by replacing all occurrences of $\ddot{\rhd}$ with $\ddot{\bowtie}$. We conclude that the statement also holds for $\circ = \ddot{\bowtie}$. □

Theorem 6.13 above does *not* generally hold for $\ddot{\lhd}$. Let $p \mathrel{\ddot{\lesseqgtr}} q \mathrel{\ddot{\lhd}} r$: intuitively, $\ddot{\lhd}$ does *not* guarantee that r's outputs will be matched by q's inputs; therefore, if p adds an input branch which is matched by an output of r, then their synchronisation may reduce to *non*-compliant behaviours, as shown in Example 6.14 below.

Example 6.14 (On $\ddot{\lesseqgtr}$ and $\ddot{\lhd}$)**.** *Consider the behaviours in Figure 6.3: p_{10} does* not *offer a* ?c*-branch matching the* !c*-branch of p_{11}; however, p_9* does *add such a branch — but if p_9 and p_{11} synchronise on it, then p_9's continuation enables a* !e*-transition which is unmatched by p_{11}'s continuation, thus violating clause* a. *of Definition 4.4. Therefore, $p_9 \mathrel{\not\ddot{\lhd}} p_{11}$.*

We stress that the situation described in Example 6.14, does *not* arise on synchronous session behaviours: this is formalised in in Proposition 6.15 below, which extends Theorem 6.13.

Proposition 6.15. *$T \mathrel{\ddot{\lesseqgtr}} U \mathrel{\ddot{\lhd}} V$ implies $T \mathrel{\ddot{\lhd}} V$.*

Proof. See page 134. Intuitively, the thesis holds because synchronous session behaviours do not have states with mixed choices among inputs and outputs, and because their shape can be determined via Theorem 6.16[2]. □

I/O simulation can be seen as a subtyping relation on general behaviours, that is $p \mathrel{\ddot{\lesseqgtr}} q$ allows p to be always used in place of q. For instance, assume that p is an asynchronous

[2]Note that the proof of Proposition 6.15 depends on Theorem 6.16, but not *vice versa*: the two results are introduced in reverse order w.r.t. their proofs for clarity of exposition.

CCS process typed with a session type q, which in turn complies with the session type r. Then, Theorem 6.13 states that I/O compliance is preserved by $\ddot{\leqslant}$, i.e. p is also I/O compliant with r, notwithstanding with the fact that p and r are specified in different calculi (actually, our statement is even more general, as it applies to *arbitrary* behaviours). Summing up, the process p will interact correctly with any process with type r (this will be formally established in Theorem 7.16).

Theorem 6.16 below states that I/O simulation is stricter than Definition 6.1. However, the two notions coincide on synchronous session behaviours. Hence, $\ddot{\leqslant}$ can be interpreted as a subtyping relation in $\mathbb{U}_{\mathrm{ST}}$, according to [GH05] (albeit in the inverse direction, corresponding to that adopted e.g. in [CHY12]).

Theorem 6.16. *(a)* $\ddot{\leqslant} \subsetneq \sqsubseteq$. *(b)* $T \mathrel{\ddot{\leqslant}} U \iff T \sqsubseteq U$.

Proof. For item (a), if $p \mathrel{\ddot{\leqslant}} q$, then by Theorem 6.13 we have that $\forall r \,.\, q \mathrel{\ddot{\bowtie}} r \implies p \mathrel{\ddot{\bowtie}} r$, i.e. $p \sqsubseteq q$ according to Definition 6.1. To prove that the inclusion is strict, recall the behaviour p_5 from Example 4.7. Since p_5 does not admit I/O compliant behaviours, it vacuously follows that $r \sqsubseteq p_5$, for all r. In particular, let r be the behaviour which loops onto itself via a $!\mathsf{a}$-transition, and assume, by contradiction, that $r \mathrel{\ddot{\leqslant}} p_5$, (say, via predictive set $\mathbb{Q}$). By clause *d.* of Definition 6.2, there must exist p' such that $\mathbb{Q} \overset{!\mathsf{a}}{\Longrightarrow} p'$ — contradiction, since $\mathbb{Q}\Downarrow^{!} \subseteq p_5\Downarrow^{!} = \emptyset$.

For item (b), the $\implies$ direction has already been proved by item (a). For the $\impliedby$ direction, let $T \sqsubseteq U$. By Lemma 6.7, we have that $T \sqsubseteq_{\mathbb{U}_{\mathrm{ST}}} U$, and so by Lemma 6.6 we can reason (up-to unfolding) by cases on the syntax T and U, to verify that (T, U) satisfies clauses *a.*–*e.* in Definition 6.2, via predictive set $\mathbb{Q} = \{U\}$. □

Remark 6.17 (On the completeness of $\ddot{\leqslant}$). *Note that the strict inclusion of Theorem 6.16 is proved with a behaviour which does* not *admit an I/O compliant one. This is not, however, a necessary condition — as shown in Figure 6.4: we have* $p_{12} \sqsubseteq p_{13} \mathrel{\ddot{\bowtie}} p_{14}$*, but* $p_{12} \mathrel{\ddot{\not\leqslant}} p_{13}$*. This kind of situation seems to only arise when dealing behaviours similar to* p_{12}*, which are impossible to obtain from (asynchronous) session types semantics: for this reason, we conjecture that* $\ddot{\leqslant}$ *and* $\sqsubseteq$ *coincide not only in* $\mathbb{U}_{ST}$ *(as shown in Theorem 6.16), but also in* $\mathbb{U}_{aST}$ *— and possibly in other suitably chosen classes of behaviours populating* $\mathbb{U}$ *(this topic will be reprised in Section 9.2).*

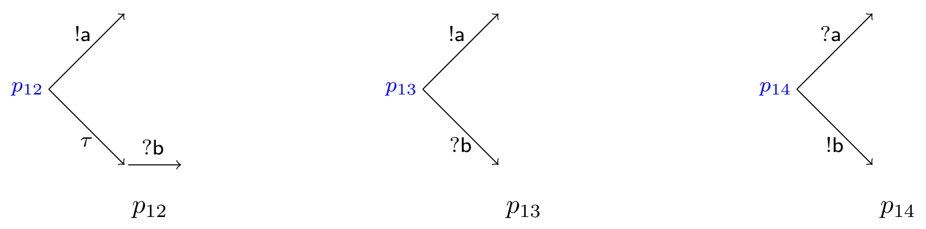


Figure 6.4: Three compliant-admitting behaviours showing that $\ddot{\leqslant} \subsetneq \sqsubseteq$.

6.4 On I/O simulation and asynchrony

In this section we study the I/O compliance relation between *asynchronous* session types, and its interplay with safety (Section 5.4). Our motivating question is: to which extent reasoning in the synchronous setting provides safety guarantees in the *a*synchronous setting?

A first guarantee is already provided by the results in the previous sections. Assume we have $T \ \ddot{\leqslant}\ U$ and $U \ \ddot{\bowtie}\ V$. Using Theorem 6.13 we can deduce $T \ \ddot{\bowtie}\ V$; then, from Proposition 4.12 we obtain $T[] \ \ddot{\bowtie}\ V[]$, and so by Theorem 5.22, we conclude that $T[] \parallel V[]$ is safe. These observations prove the following proposition.

Proposition 6.18. *If* $T \ \ddot{\leqslant}\ U \ \ddot{\bowtie}\ V$*, then* $T[] \parallel V[]$ *is safe.*

Note that Proposition 6.18 requires knowing that U is compliant with V in the *synchronous* setting. In the rest of this section, we aim at strengthening this results, by weakening this assumption and just require that U and V are compliant in the *asynchronous* setting. To do that, we focus on the following question:

$$\text{if } T \ \ddot{\leqslant}\ U, \ \text{ does } T[] \ \ddot{\leqslant}\ U[] \text{ hold?}$$

Indeed, if the answer is positive, then whenever $U[] \ \ddot{\bowtie}\ V[]$, we can replace $U[]$ with $T[]$ and obtain a safe system $T[] \parallel V[]$. This would allow to refine asynchronous session behaviours by only reasoning on their *synchronous* semantics (i.e., proving $T \ \ddot{\leqslant}\ U$).

A first result is that the answer to such a question is negative in general, as shown in the following example.

Example 6.19. *Consider the following session types:*

$$T = \mathrm{rec}_X\, !\mathsf{a}.X \qquad U = \mathrm{rec}_Y\, !\mathsf{a}.X \oplus !\mathsf{b}.?\mathsf{c}$$

We have $T \mathrel{\ddot{\leqslant}} U$ *but* $T[] \mathrel{\ddot{\not\leqslant}} U[]$*, because clause* b. *of Definition* 6.2 *is violated.*

We can prove the last statement by contradiction: assume that $T[] \mathrel{\ddot{\leqslant}} U[]$*, with some predictive set* $\mathbb{U}$*. Since* $U[]\xRightarrow{??\mathsf{c}}$*, then by Proposition* C.6 *we must have* $\mathbb{U}\xRightarrow{??\mathsf{c}}$*. Then, by the first part of clause* b. *in Definition* 6.2*, it must be* $?\mathsf{c} \in T[]\Downarrow^{?}$ *— contradiction (since* $T[]\Downarrow^{?} = \emptyset$*).*

Example 6.19 above can also be used to provide a negative answer to the following question:

$$\text{if } T \mathrel{\ddot{\leqslant}} U \text{ and } U[] \mathrel{\ddot{\bowtie}} V[], \text{ does } T[] \mathrel{\ddot{\bowtie}} V[] \text{ hold?}$$

as shown in the following example.

Example 6.20. *Consider* T, U *from Example* 6.19*, and let:*

$$V \;=\; !\mathsf{c}.\mathrm{rec}_Y\, ?\mathsf{a}.Y \,\&\, ?\mathsf{b}$$

Note that $U[] \mathrel{\ddot{\bowtie}} V[]$*, albeit* $U \mathrel{\ddot{\not\bowtie}} V$*. However,* $T[] \mathrel{\ddot{\not\bowtie}} V[]$*: in fact, we have* $V[]\xRightarrow{!\mathsf{c}}$ *but* $T[]\overset{?\mathsf{c}}{\not\Rightarrow}$*, thus violating clause* a. *of Definition* 4.4*. Moreover,* $T[] \parallel V[]$ *is* not *safe: in fact, according to Definition* 5.5*, it is an orphan message configuration, with* $!\mathsf{c}$ *orphan message of* $V[]$*.*

The previous example implies that also the following question has a negative answer:

$$\text{if } T \sqsubseteq U, \text{ does } T[] \sqsubseteq U[] \text{ hold?}$$

Indeed, by choosing T and U as in Example 6.20, we have that $T \sqsubseteq U$ (by item (a) of Theorem 6.16), but $T[] \not\sqsubseteq U[]$ as shown in the example.

Intuitively, the non-preservation of $\ddot{\leqslant}$ and $\sqsubseteq$ in the asynchronous setting is due to the fact that, in the asynchronous semantics, output prefixes are turned into τ-prefixes; therefore, inputs that are reachable after a sequence of outputs in some session type T become weakly reachable in $T[\sigma]$ (this notion will be formalised later, in Proposition 6.25), and thus relevant for clause *a.* of Definition 4.4. Moreover, depending on the branching structure of T, such inputs may also become weakly persistent (as per Definition 2.6), and thus relevant for clauses *b.* and *e.* of Definition 6.2.

Following this intuition, we provide in Definition 6.21 below a sufficient condition which guarantees the preservation of $\ddot{\leqslant}$ when passing from the synchronous to the asynchronous semantics.

Definition 6.21 (Input-preserving behaviours)**.** p is input-preserving w.r.t. q *(in symbols: $p \preceq_{?} q$) iff for all $w = \ell_1, \ldots, \ell_n, p', q', \mathsf{a}$, whenever*

$$p \overset{w}{\Rightarrow} p' \;\textit{ and }\; q \overset{w}{\Rightarrow} q' \tag{6.12a}$$

$$\textit{and} \quad \left(\forall q'' \,.\, q' \overset{!}{\Rightarrow}{}^{*} q'' \textit{ implies } q'' \overset{!}{\Rightarrow}{}^{*} \xrightarrow{?\mathsf{a}}\right), \tag{6.12b}$$

$$\textit{then } p' \overset{!}{\Rightarrow}{}^{*} \xrightarrow{?\mathsf{a}} . \tag{6.12c}$$

Intuitively, Definition 6.21 says that if $p \preceq_{?} q$, then whenever p and q reach some states p', q' through the same weak transitions (item 6.12a) and q' *always* exposes some input ?a (possibly after some outputs, by item 6.12b), then p' cannot "forget" ?a, and must be able to expose it (possibly after some outputs). Or, in other words: if ?a is "persistent" (albeit through some outputs) on the RHS, then it must also be reachable (possibly after some outputs) on the LHS. Notice that the direction of $\preceq_{?}$ follows the intuition of the contravariance of inputs, as in $\ddot{\leqslant}$.

Example 6.22. *Consider T and U from Example 6.19: we have $T \not\preceq_{?} U$. Instead, as positive examples, we have:*

a. $!\mathsf{a}.!\mathsf{a}.!\mathsf{b}.?\mathsf{c}$ $(\ddot{\leqslant} \cap \preceq_{?})$ U *(i.e., the LHS can select the* !a*-branch of U for a finite number of times);*

b. $\mathrm{rec}_X\, !\mathsf{a}.?\mathsf{c}'.X$ $(\ddot{\leqslant} \cap \preceq_{?})$ $\mathrm{rec}_Y\, !\mathsf{a}.?\mathsf{c}'.Y \oplus !\mathsf{b}.?\mathsf{c}$ *(i.e., the LHS can infinitely often select a recursive branch that always performs an input).*

The relation holds vacuously e.g. in $!\mathsf{a}.?\mathsf{c} \preceq_{?} !\mathsf{b}.?\mathsf{c}' \oplus !\mathsf{b}'$ (but in this case, $\ddot{\leqslant}$ does not hold).

Definition 6.21 is used in Theorem 6.27 below, which generalises Theorem 4.13 and proposition 4.12, extending to I/O simulation the set of properties preserved when passing from synchronous to asynchronous semantics. This result is based on four key properties:

(i) whenever $p \;\ddot{\leqslant}\; q$, each weak sequence of outputs of p is I/O simulated by q (Proposition 6.23);

(ii) whenever $p \preceq_{?} q$, and the two behaviours reach p', q' through the same weak transitions, then $p' \preceq_{?} q'$ (Proposition 6.24);

(iii) each weak sequence of ouptuts from a session behaviour T corresponds to a sequence of τs in its asynchronous version $T[\sigma]$, for all σ (Proposition 6.25);

(iv) *vice versa*, each sequence of τs from $T[\sigma]$ corresponds to a weak sequence of outputs from T (Proposition 6.26).

Proposition 6.23. *Let $p \mathrel{\ddot{\leqslant}} q$. Then, for all $w \in (\mathsf{A}^!)^*$, $p \overset{w}{\Rightarrow} p'$ implies $\exists q' \,.\, q \overset{w}{\Rightarrow} q'$ and $p' \mathrel{\ddot{\leqslant}} q'$.*

Proof. By induction on w. In the base case (when w is empty) we need to prove that $p \Rightarrow p'$ implies $\exists q' \,.\, q \Rightarrow q'$ and $p' \mathrel{\ddot{\leqslant}} q'$: this follows from Lemma C.2, letting $q' = q$.

For the inductive step, let $w = w'!\mathsf{a}$ (where w' is a sequence of outputs). We have $p \overset{w'}{\Longrightarrow} p''$ and $q \overset{w'}{\Longrightarrow} q''$, with $p'' \mathrel{\ddot{\leqslant}} q''$ (by the induction hypothesis); we need to show that:

$$\forall p_0, p_1 \,.\, p'' \Rightarrow p_0 \xrightarrow{!\mathsf{a}} p_1 \Rightarrow p' \quad \text{implies} \quad \exists q' \,.\, q'' \overset{!\mathsf{a}}{\Rightarrow} q' \text{ and } p' \mathrel{\ddot{\leqslant}} q'$$

Now, by Lemma C.2, we have $p_0 \mathrel{\ddot{\leqslant}} q''$; by clause *d.* of Definition 6.2, we have that for some predictive set $\mathbb{Q}$, $p_0 \xrightarrow{!\mathsf{a}} p_1$ implies $\exists q''' \,.\, \mathbb{Q} \overset{!\mathsf{a}}{\Rightarrow} q'''$ and $p_1 \mathrel{\ddot{\leqslant}} q'''$; since (by Definition 6.2) $q'' \Rrightarrow \mathbb{Q}$, we have $q'' \overset{!\mathsf{a}}{\Rightarrow} q'''$; finally, by Lemma C.2, we obtain $p' \mathrel{\ddot{\leqslant}} q'''$. We conclude by letting $q' = q'''$, thus obtaining the thesis. □

Proposition 6.24. *Let $p \preceq_{?} q$. Then, $\forall w \,.\, p \overset{w}{\Rightarrow} p'$ and $q \overset{w}{\Rightarrow} q'$ implies $p' \preceq_{?} q'$.*

Proof. We prove the contrapositive. We have that $\exists w \,.\, p \overset{w}{\Rightarrow} p'$ and $q \overset{w}{\Rightarrow} q'$ but $p' \not\preceq_{?} q'$. This means that $\exists w', p''', q''', \mathsf{a}$ such that:

- $p' \overset{w'}{\Longrightarrow} p'''$ and $q' \overset{w'}{\Longrightarrow} q'''$ (item 6.12a of Definition 6.21), and
- $\forall q'' \,.\, q''' \overset{!}{\Rightarrow}^* q''$ implies $q'' \overset{!}{\Rightarrow}^* \xrightarrow{?\mathsf{a}}$ (item 6.12b of Definition 6.21),

and $p''' \overset{!}{\Rightarrow}^* \xrightarrow{?\mathsf{a}}$. is false. Then, under the same existential quantifications, we also have:

- $p \overset{ww'}{\Longrightarrow} p'''$ and $q \overset{ww'}{\Longrightarrow} q'''$ (item 6.12a of Definition 6.21), and
- $\forall q'' \,.\, q''' \overset{!}{\Rightarrow}^* q''$ implies $q'' \overset{!}{\Rightarrow}^* \xrightarrow{?\mathsf{a}}$ (item 6.12b of Definition 6.21),

and $p''' \overset{!}{\Rightarrow}{}^* \xrightarrow{?\mathsf{a}}$. is false. Therefore, we conclude $p \not\precsim_? q$. ☐

Proposition 6.25. *For all* $\sigma, T, T', w \in (\mathsf{A}^!)^*$*:* $T \overset{w}{\Rightarrow} T'$ *implies* $\exists \sigma' \,.\, T[\sigma] \Rightarrow T'[\sigma']$.

Proof. We proceed by induction on the length of the sequence of transitions in $T \overset{w}{\Rightarrow} T'$. The base case is trivial: since we have no transitions, then $T' = T$ and we conclude by letting $\sigma' = \sigma$. For the inductive cases, we have:

- $T \overset{w}{\Rightarrow} T'' \xrightarrow{\tau} T'$. By the induction hypothesis, we have $\exists \sigma'' \,.\, T[\sigma] \Rightarrow T''[\sigma'']$. By item *(ii)* of Proposition 2.19, $\exists \mathsf{b} \,.\, T''[\sigma''] \xrightarrow{\tau} T'[\sigma''.!\mathsf{b}]$. We conclude by letting $\sigma' = \sigma''.!\mathsf{b}$;
- $T \overset{w'}{\Rightarrow} T'' \xrightarrow{!\mathsf{b}} T'$. By the induction hypothesis, we have $\exists \sigma'' \,.\, T[\sigma] \Rightarrow T''[\sigma'']$. By item *(iii)* of Proposition 2.19, $T''[\sigma''] \xrightarrow{\tau} T'[\sigma''.!\mathsf{b}]$. We conclude by letting $\sigma' = \sigma''.!\mathsf{b}$.

☐

Proposition 6.26. *If* $T[\sigma] \Rightarrow T'[\sigma']$*, then* $\exists w \in (\mathsf{A}^!)^* \,.\, T \overset{w}{\Rightarrow} T'$.

Proof. We proceed by induction on the length of the sequence of transitions in $T[\sigma] \Rightarrow T'[\sigma']$. The base case is trivial: since we have no transitions, then $T' = T$ and $\sigma' = \sigma$ — and we conclude by letting w be the empty sequence. For the inductive case, let $T[\sigma] \Rightarrow T''[\sigma''] \xrightarrow{\tau} T'[\sigma']$. By the induction hypothesis, there exists a sequence of outputs w' such that $T \overset{w'}{\Rightarrow} T''$. Now, from $T''[\sigma''] \xrightarrow{\tau} T'[\sigma']$, by Definition 2.16 we have that T'' is equivalent to a non-empty internal choice with some $!\mathsf{b}$-guarded branch, and $\sigma' = \sigma''.!\mathsf{b}$; moreover, by Proposition 2.19 (items *(ii)* and *(iii)*), we have either $T'' \xrightarrow{\tau}\xrightarrow{!\mathsf{b}} T'$ or $T \xrightarrow{!\mathsf{b}} T'$ — and therefore, $T'' \overset{!\mathsf{b}}{\Rightarrow} T'$. Hence, from $T \overset{w'}{\Rightarrow} T'' \overset{!\mathsf{b}}{\Rightarrow} T'$, we conclude by letting $w = w'!\mathsf{b}$. ☐

Theorem 6.27. *If* $T \;(\ddot{\lessapprox} \cap \preceq_?)\; U$*, then* $T[] \;\ddot{\lessapprox}\; U[]$.

Proof. Let us define:

$$\ddot{\mathfrak{R}} \;=\; \left\{ (T[\sigma]\,,\, U[\sigma]) \;\middle|\; T\;(\ddot{\lessapprox} \cap \preceq_?)\; U \right\}$$

We show that $\ddot{\mathfrak{R}}$ is an I/O simulation. For each $(T[\sigma]\,,\, U[\sigma]) \in \ddot{\mathfrak{R}}$, we define $\mathbb{U} = \{U[\sigma]\}$ as predictive set, and verify the clauses of Definition 6.2:

item *a.* assume $T[\sigma]{\Downarrow}^{!} = \emptyset$. Then, by Definition 2.16 we have $\sigma = \epsilon$, and T can only be a (possibly empty) external choice. Since $T \ddot{\leqslant} U$, we have that U is also an external choice (by Theorem 6.16, Lemma 6.7 and Lemma 6.6) — and thus, since $\mathbb{U} = \{U[\sigma]\}$, we conclude $\mathbb{U}{\Downarrow}^{!} = \emptyset$;

item *b.* since $T \ddot{\leqslant} U$, from Theorem 6.16, Lemma 6.7 and Lemma 6.6 we can determine that (up-to unfolding) T and U can be either:

- both empty choices, i.e. $T = U = \mathbf{0}$. Then, since $\mathbb{U} = \{U[\sigma]\}$, we have $\emptyset = \mathbb{U}{\Downarrow}^{??} \subseteq T[\sigma]{\Downarrow}^{?} = \emptyset$ and $\left(\mathbb{U}{\Downarrow}^{?} = \emptyset \implies T[\sigma]{\Downarrow}^{?} = \emptyset\right)$;
- both *non*-empty external choices, with all branches of U included in T. We notice that, by Proposition 2.19 (item *(i)*), we have:
  - $\forall \mathsf{a} \,.\, T \xrightarrow{?\mathsf{a}}$ iff $T[\sigma] \xrightarrow{?\mathsf{a}}$, and
  - $\forall \mathsf{a} \,.\, U \xrightarrow{?\mathsf{a}}$ iff $U[\sigma] \xrightarrow{?\mathsf{a}}$.

  Therefore, $\forall \mathsf{a} . U[\sigma] \xrightarrow{?\mathsf{a}}$ implies $T[\sigma] \xrightarrow{?\mathsf{a}}$. We also notice that, by Definition 2.16, we have:
  - $\forall \mathsf{a} \,.\, U[\sigma] \xrightarrow{?\mathsf{a}}$ iff $U[\sigma] \xRightarrow{??\mathsf{a}}$ (i.e., all inputs of external choices are persistent);

  - $\forall \mathsf{a} \,.\, T[\sigma] \xRightarrow{?\mathsf{a}}$ iff $T[\sigma] \xrightarrow{?\mathsf{a}}$ (i.e., all weakly reachable inputs are also immediately enabled).

  Summing up, $\forall \mathsf{a}$ we have:

$$\begin{tikzcd}
U[\sigma] \xRightarrow{??\mathsf{a}} \arrow[r, phantom, "\iff"] & U[\sigma] \xrightarrow{?\mathsf{a}} \arrow[r, phantom, "\iff"] & U \xrightarrow{?\mathsf{a}} \arrow[d, Rightarrow] \\
T[\sigma] \xRightarrow{?\mathsf{a}} \arrow[r, phantom, "\iff"] & T[\sigma] \xrightarrow{?\mathsf{a}} \arrow[r, phantom, "\iff"] & T \xrightarrow{?\mathsf{a}}
\end{tikzcd}$$

  and therefore, since $\mathbb{U} = \{U[\sigma]\}$,

$$U[\sigma]{\Downarrow}^{??} = \mathbb{U}{\Downarrow}^{??} \subseteq T[\sigma]{\Downarrow}^{?}$$

  Furthermore, we also have that $\mathbb{U}{\Downarrow}^{?} = \emptyset$ implies $T[\sigma]{\Downarrow}^{?} = \emptyset$ (vacuously);
- both *non*-empty internal choices. We first prove $\mathbb{U}{\Downarrow}^{??} \subseteq T[\sigma]{\Downarrow}^{?}$. If $\mathbb{U}{\Downarrow}^{??} = \emptyset$, the thesis is immediate. Otherwise, assume $\exists \mathsf{a} \,.\, \mathbb{U} \xRightarrow{??\mathsf{a}}$; since $\mathbb{U} = \{U[\sigma]\}$, this holds iff $U[\sigma] \xRightarrow{??\mathsf{a}}$ — and by Definition 2.6, this means:

$$\forall U', \sigma' \,.\, U[\sigma] \Rightarrow U'[\sigma'] \;\text{ implies }\; \exists U'', \sigma'' \,.\, U'[\sigma'] \Rightarrow U''[\sigma''] \xrightarrow{?\mathsf{a}}$$

By Proposition 6.26, each τ-sequence between $U[\sigma]$ and $U'[\sigma']$ above corresponds to some sequence of outputs w such that $U \overset{w}{\Rightarrow} U'$; similarly, the τ-sequence between $U'[\sigma']$ and $U''[\sigma'']$ above corresponds to some sequence of outputs w' such that $U' \overset{w'}{\Longrightarrow} U''$. Thus,

$$\forall U', w = !\mathsf{b}_1, \ldots, !\mathsf{b}_n \,.\, U \overset{w}{\Rightarrow} U' \ \text{ implies } \ \exists U'', w' = !\mathsf{c}_1, \ldots, !\mathsf{c}_{n'} \,.\, U' \overset{w'}{\Rightarrow} U'' \xrightarrow{?\mathsf{a}}$$

and therefore, by Notation 2.1,

$$\forall U' \,.\, U \overset{!}{\Rightarrow}{}^{*} U' \ \text{ implies } \ \exists U'' \,.\, U' \overset{!}{\Rightarrow}{}^{*} U'' \xrightarrow{?\mathsf{a}} \tag{6.13}$$

Now, since $T \preceq_{?} U$, we have:

- an *empty* sequence of actions w_0 matches item 6.12a of Definition 6.21 for $T \overset{w_0}{\Longrightarrow} T$ and $U \overset{w_0}{\Longrightarrow} U$;
- from Equation (6.13), we have that $?\mathsf{a}$ is "persistent" in U by item 6.12b of Definition 6.21;
- therefore, by item 6.12c of Definition 6.21, $\exists T' \,.\, T \overset{!}{\Rightarrow}{}^{*} T' \xrightarrow{?\mathsf{a}}$.

Therefore, $\exists w'' = !\mathsf{d}_1, \ldots, !\mathsf{d}_m$ such that $T \overset{w''}{\Longrightarrow} T' \xrightarrow{?\mathsf{a}}$; and then, by Proposition 6.25, we have $\exists \sigma' \,.\, T[\sigma] \Rightarrow T'[\sigma'] \xrightarrow{?\mathsf{a}}$. Summing up, we have shown that $\forall \mathsf{a}$, $\mathbb{U} \overset{??\mathsf{a}}{\Longrightarrow}$ implies $T[\sigma] \overset{?\mathsf{a}}{\Longrightarrow}$: we conclude $\mathbb{U}\Downarrow^{??} \subseteq T[\sigma]\Downarrow^{?}$.

We are left to prove that $\mathbb{U}\Downarrow^{?} = \emptyset$ implies $T[\sigma]\Downarrow^{?} = \emptyset$. We first observe that since $\mathbb{U} = \{U[\sigma]\}$, then $\mathbb{U}\Downarrow^{?} = \emptyset$ implies $U[\sigma]\Downarrow^{?} = \emptyset$, which in turn gives us

$$\forall U', \sigma'' \,.\, U[\sigma] \Rightarrow U'[\sigma''] \ \text{ implies } \ U'[\sigma'']\Downarrow^{?} = \emptyset \tag{6.14}$$

Let us now examine the possible weak transitions of $T[\sigma]$. From Proposition 6.26 we know that $\forall T', \sigma' . T[\sigma] \Rightarrow T'[\sigma']$ implies $\exists w = !\mathsf{b}_1, \ldots, !\mathsf{b}_n . T \overset{w}{\Rightarrow} T'$. Moreover, since $T \mathrel{\ddot{\lessgtr}} U$, by Proposition 6.23 we have that for all such w, $\exists U' \,.\, U \overset{w}{\Rightarrow} U'$ and $T' \mathrel{\ddot{\lessgtr}} U'$; and by Proposition 6.25, $U \overset{w}{\Rightarrow} U'$ implies $\exists \sigma'' \,.\, U[\sigma] \Rightarrow U'[\sigma'']$. From this, Equation (6.14), gives $U'[\sigma'']\Downarrow^{?} = \emptyset$ — i.e., by item *(i)* of Proposition 2.19, $U'\Downarrow^{?} = \emptyset$. But then, $T' \mathrel{\ddot{\lessgtr}} U'$ is supported by some predictive set $\mathbb{U}'$ such that $\mathbb{U}'\Downarrow^{?} = \emptyset$; and therefore, by item *b.* of Definition 6.2, we have $T'\Downarrow^{?} = \emptyset$ — and by item *(i)* of Proposition 2.19, $T'[\sigma']\Downarrow^{?} = \emptyset$. Thus, we conclude $T[\sigma]\Downarrow^{?} = \emptyset$.

item *c.* assume $T[\sigma] \xrightarrow{\tau} T'[\sigma']$. Notice that, by Definition 2.16, $T[\sigma]$ can only generate a τ-transition when T is equivalent to an internal choice with some $\mathsf{!a}$-branch, and $\mathsf{!a}$ is appended to σ: hence, $\sigma' = \sigma\,.\mathsf{!a}$. Then, by Definition 2.15, we have $T \Rightarrow\xrightarrow{\mathsf{!a}} T'$. Since $T \ddot{\leqslant} U$, by Proposition 6.23 we have $\exists U'', U', U'''\,.\,U \Rightarrow U'' \xrightarrow{\mathsf{!a}} U' \Rightarrow U'''$ and $T' \ddot{\leqslant} U'''$. By Proposition 2.18, U'' is a single-branch internal choice, and by item *(iii)* of Proposition 2.19, $U[\sigma] \xrightarrow{\tau} U'[\sigma\,.\mathsf{!a}] = U'[\sigma']$. Moreover, by Lemma C.3 we have $T' \ddot{\leqslant} U'$, and by Proposition 6.24 we have $T' \preceq_? U'$. Therefore, from the definition of $\mathbb{U}$ above, we conclude $\exists U'\,.\,\mathbb{U} \Rightarrow U'[\sigma'] \land T'[\sigma']\ \ddot{\mathfrak{R}}\ U'[\sigma']$;

item *d.* assume $T[\sigma] \xrightarrow{\mathsf{!a}} T'[\sigma']$. Notice that, by the semantics in Definition 2.16, $T[\sigma]$ can only generate a $\mathsf{!a}$-transition when an output is removed from the head of σ, turning it into σ', without changing T: so, $T' = T$. The same observation holds for $U[\sigma]$. Since $\mathbb{U} = \{U\}$, we have $\exists U'\,.\,\mathbb{U} \overset{\mathsf{!a}}{\Rightarrow} U'[\sigma']$ with $U' = U$, from which we obtain $T = T' \ddot{\leqslant} U'$; moreover, by Proposition 6.24 we have $T' \preceq_? U'$: we conclude $T'[\sigma']\ \ddot{\mathfrak{R}}\ U'[\sigma']$;

item *e.* assume $T[\sigma] \xrightarrow{\mathsf{?a}} T'[\sigma'] \land \mathbb{U}\overset{\mathsf{??a}}{\Longrightarrow}$. Notice that, by the semantics in Definition 2.16, $T[\sigma]$ can only generate a $\mathsf{?a}$-transition when T is an external choice, and σ is unchanged by the reduction — i.e., $\sigma' = \sigma$. Now, since $T \ddot{\leqslant} U$, by Theorem 6.16, Lemma 6.7 and Lemma 6.6 we have that U is an external choice, too — and by Definition 2.15, we can verify that $U\xrightarrow{\mathsf{?a}}$ iff $U\overset{\mathsf{??a}}{\Longrightarrow}$, and since U is an external choice, by Definition 2.16 we have $U[\sigma]\xrightarrow{\mathsf{?a}}$ implies $U[\sigma]\overset{\mathsf{??a}}{\Longrightarrow}$. Moreover, since $T \ddot{\leqslant} U$, again by Theorem 6.16, Lemma 6.7 and Lemma 6.6 we also have $\forall U'\,.\,T \xrightarrow{\mathsf{?a}} T' \land U \xrightarrow{\mathsf{?a}} U'$ implies $T' \ddot{\leqslant} U'$. Finally, we observe that by Proposition 6.24, $\forall U'\,.\,T \xrightarrow{\mathsf{?a}} T' \land U \xrightarrow{\mathsf{?a}} U'$ implies $T' \preceq_? U'$. Therefore, by definition of $\mathfrak{R}$, we conclude $\exists U'\,.\,\mathbb{U} \overset{\mathsf{?a}}{\Rightarrow} U'[\sigma'] \land T'[\sigma']\ \ddot{\mathfrak{R}}\ U'[\sigma']$.

Hence, $\ddot{\mathfrak{R}}$ is an I/O simulation. Thus, $\forall\sigma$, $T\ (\ddot{\leqslant} \cap \preceq_?)\ U$ implies $T[\sigma]\ \ddot{\mathfrak{R}}\ U[\sigma]$, and therefore, $T[\sigma]\ \ddot{\leqslant}\ U[\sigma]$. We conclude $T[]\ \ddot{\leqslant}\ U[]$. □

Indeed, the above theorem together with Proposition 6.18 give us some general criterions for proving properties of asynchronous session types: it suffices to show them in the simpler synchronous case (which is finite-state by Definition 2.15, and therefore guarantees the decidability of all the relations mentioned in Theorem 6.27).

Chapter 7

Session types without types

In this chapter we address the topics discussed in Sections 1.1.1 and 1.1.2: we show how typical (and not-so-typical) session typing rules can arise by applying I/O compliance and I/O simulation on specific process and type languages — in our case, sync/async CCS and session types from Chapter 2.

7.1 From semantics to syntax

Our treatment so far does not depend on a syntactic representation of behaviours in $\mathbb{U}$. In the resulting unifying view, there are no inherent distinctions between processes and types: they are just behaviours in an LTS. This allows us to define relations between objects which morally belong to different realms: e.g. $p \mathrel{\ddot{\leqslant}} q$ may relate, say, an async CCS process with a (synchronous or asynchronous) session type.

The price for this generalisation is (seemingly) the loss of a useful feature: using syntax-based reasoning to check whether a process has a certain type, without having to deal with the semantic level. In this chapter, we show how this possibility can be restored in four steps:

(i) choosing a process language and a type language (with their corresponding semantics);

(ii) encoding types into processes;

(iii) devising a sound set of axioms and rules for $\ddot{\leqslant}$;

$$\begin{array}{rclcrcl}
P & \ddot{\approx} & P + \mathbf{0} & \qquad & P & \ddot{\approx} & P \mid \mathbf{0} \\
P + Q & \ddot{\approx} & Q + P & & P \mid Q & \ddot{\approx} & Q \mid P \\
P + (Q + R) & \ddot{\approx} & (P + Q) + R & & P \mid (Q \mid R) & \ddot{\approx} & (P \mid Q) \mid R \\
P & \ddot{\approx} & P + P & & P & \ddot{\approx} & P
\end{array}$$

Table 7.1: Some axioms for $\ddot{\approx}$ in $\mathbb{U}_{\text{CCS}}$ and $\mathbb{U}_{\text{aCCS}}$ (where $\ddot{\approx}$ is the largest symmetric I/O simulation, by Definition 6.2).

(iv) using these axioms to induce syntax-based typing rules that imply (i.e., safely approximate) $\ddot{\lessgtr}$.

We give a proof-of-concept of this methodology: for step *(i)* above, we focus on async CCS ($\mathbb{U}_{\text{aCCS}}$) for processes[1], and async session behaviours ($\mathbb{U}_{\text{aST}}$) as types.

Remark 7.1. *To improve readability, hereafter we shall sometimes omit the buffers $[\sigma]$ appearing in $\mathbb{U}_{aCCS}$ processes. Moreover, we will write $P \,\ddot{\lessgtr}\, Q$ in $\mathbb{U}_{aCCS}$ with the meaning: $\forall\sigma \,.\, P[\sigma] \,\ddot{\lessgtr}\, Q[\sigma]$.*

The encoding from types to processes for step *(ii)* is the one given in Definition 2.27.

Proceeding to step *(iii)*, we now aim at a $\ddot{\lessgtr}$-based inductive relation for $\mathbb{U}_{\text{aCCS}}$. We start by introducing a syntax-based equivalence $\equiv$ between $\mathbb{U}_{\text{CCS}}$ and $\mathbb{U}_{\text{aCCS}}$ processes[2], in Definition 7.3 below. As expected, $\equiv$ embodies the commutative monoidal laws for $+$ and $\mid$ (with $\mathbf{0}$ as neutral element), the absorption law for $+$, and reflexivity: here, we semantically ground $\equiv$ on the I/O bisimilarity of the LHS and RHS, by defining $\equiv$ upon a set of axioms for $\ddot{\approx}$, listed in Table 7.1.

Proposition 7.2. *The relations in Table 7.1 hold for all $\mathbb{U}_{aCCS}$ processes.*

Proof. We can verify that the transition diagrams of the LHS and RHS of each relation are isomorphic, both synchronously (by Definition 2.24) and asynchronously (by Definition 2.25, when a buffer $[\sigma]$ is added to both processes). Therefore, they are also (strongly and weakly) bisimilar: we conclude by Theorem 6.11. $\square$

[1]Coherently with the abstractions described in Section 1.2, our processes are not as rich as the variants of π-calculus usually studied in session types literature: intuitively, we focus on the interactions that a process performs *inside* a session, while every other activity is abstracted as a τ-move. These choices will be further discussed in Sections 9.3 and 9.8.

[2]Here, we save some notation by overloading the symbol $\equiv$, which was already introduced as a relation between session types in Definition A.1. Despite the possible ambiguity, the context should always allow to determine which relation is being used.

$$\frac{\forall i \in I \,.\, P_i \mathrel{\ddot{\leqslant}} Q_i}{\sum_{i\in I} \ell_{\tau i} \,.\, P_i \mathrel{\ddot{\leqslant}} \sum_{i\in I} \ell_{\tau i} \,.\, Q_i}\ (+\textsc{Ctx}) \qquad \frac{\begin{array}{c} Q \equiv \sum_{i\in I} !\mathsf{a}_i \,.\, Q_i \\ \forall i \in I \,.\, P_i \mathrel{\ddot{\leqslant}} Q_i \qquad I \neq \emptyset \end{array}}{\sum_{i\in I} !\mathsf{a}_i \,.\, P_i \mathrel{\ddot{\leqslant}} Q + !\mathsf{b} \,.\, Q'}\ (+\textsc{Int})$$

$$\frac{\begin{array}{ccc} Q \equiv \sum_{i\in I} ?\mathsf{a}_i \,.\, Q_i & & \forall k \in K \,.\, P_k \mathrel{\ddot{\leqslant}} Q \\ \forall i \in I \,.\, P_i \mathrel{\ddot{\leqslant}} Q_i & \emptyset \neq I \subseteq J & \forall j \in J \setminus I \,.\, \mathsf{a}_j \notin \{\mathsf{a}_i\}_{i\in I} \end{array}}{\sum_{j\in J} (?\mathsf{a}_j \,.\, P_j) + \sum_{k\in K} \tau \,.\, P_k \mathrel{\ddot{\leqslant}} Q}\ (+\textsc{Ext})$$

$$\frac{\forall i \in I \,.\, P_i \mathrel{\ddot{\leqslant}} Q}{\sum_{i\in I} \tau \,.\, P_i \mathrel{\ddot{\leqslant}} Q}\ (+\tau) \qquad \frac{P \mathrel{\ddot{\leqslant}} Q \quad R \mathrel{\ddot{\leqslant}} S \quad \mathrm{ins}(P) = \mathrm{ins}(Q) = \emptyset}{P \mid R \mathrel{\ddot{\leqslant}} Q \mid S}\ (|\mathrm{LR})$$

- -

$$\frac{\begin{array}{c} Q \equiv \sum_{i\in I} !\mathsf{c}_i \,.\, Q_i \\ P' \mathrel{\ddot{\leqslant}} !\mathsf{a} \quad P'' \mathrel{\ddot{\leqslant}} ?\mathsf{b} \end{array}}{P' \mid P'' \mathrel{\ddot{\leqslant}} !\mathsf{a} \,.\, ?\mathsf{b} + Q}\ (|\mathrm{L}) \qquad \frac{P \mathrel{\ddot{\leqslant}} ?\mathsf{b} \,.\, Q \quad R \equiv \sum_{i\in I} !\mathsf{c}_i \,.\, R_i}{!\mathsf{a} \,.\, P \mathrel{\ddot{\leqslant}} !\mathsf{a} + R \mid ?\mathsf{b} \,.\, Q}\ (|\mathrm{R})$$

Table 7.2: Rules for $\ddot{\leqslant}$ in $\mathbb{U}_{\mathrm{aCCS}}$. $\mathrm{ins}(P)$ gives the set of inputs appearing in P's body. Note that the rules above the dashed line are also valid for $\mathbb{U}_{\mathrm{CCS}}$.

Definition 7.3. $\equiv$ *is the least symmetric and transitive relation between* $\mathbb{U}_{CCS}$ *terms, and between* $\mathbb{U}_{aCCS}$ *terms, satisfying the axioms in Table 7.1 by replacing the occurrences of* $\ddot{\approx}$ *with* $\equiv$.

Proposition 7.4. $\equiv$ *(from Definition 7.3) is an equivalence relation.*

Proof. Follows from Definition 7.3: symmetry and transitivity are immediate; reflexivity holds by the identity rule in Table 7.1. □

We will now follow a similar approach to define a syntax-based relation which is semantically grounded on $\ddot{\leqslant}$: we will introduce a set of rules and later use them under and inductive interpretation.

Such rules are listed in Table 7.2, and they are mostly straightforward:

- $(+\textsc{Ctx})$ says that each P_i which is I/O simulated by some Q_i can replace the latter in the context of a guarded choice;
- $(+\textsc{Int})$ and $(+\textsc{Ext})$ (with $K = \emptyset$) correspond to the typical session typing rules for "pure" internal/external choices (resp. with outputs and inputs), allowing to add inputs and remove outputs on the LHS, according to $\ddot{\leqslant}$;

- (+Ext) with $K \neq \emptyset$ handles an external choice that is interrupted (with τ-moves) and later reprised: this is a simple case of Erlang-style `receive...after...` behaviour, as seen in Chapter 3;
- ($+\tau$) allows the LHS to internally choose one continuation among P_i, provided that each one is I/O simulated by Q;
- (|LR) allows the parallel composition of behaviours. The side condition requires that one of the parallel components does not contain input actions: this ensures that the two threads cannot interfere badly (i.e., introduce additional non-determinism by exposing the same inputs) along their reductions;
- (|L) exploits the asynchronous semantics of $\mathbb{U}_{\mathrm{aCCS}}$: it allows to represent an output and an input in parallel, when they are syntactically sequential in the process on the RHS (which is required to be an output-guarded choice);
- (|R), on the opposite, allows the process on the LHS to be the sequential composition $\mathsf{!a}\,.\,P$ when a parallel composition appears on the RHS: the conditions are that P is I/O simulated by an external choice, and that $\mathsf{!a}$ guards a terminating branch of an internal choice.

Lemma 7.5 establishes the correctness of the rules above.

Lemma 7.5 (Rules for $\ddot{\leqslant}$)**.** *For all $\mathbb{U}_{aCCS}$ processes, the rules in Table 7.2 hold.*

Proof. See page 138. □

We can now use the rules in Table 7.2 to construct a typing system with *inductive* syntax-based rules. A first suggestion of what we are aiming to is given by Example 7.6 below.

Example 7.6. *From Chapter 3, recall Alice's type T''_{A} and process P''_{A} when she is late for work.*

$$T''_{\mathsf{A}} \;=\; \mathsf{!aCoffee.!pay.?coffee} \qquad\qquad P''_{\mathsf{A}} \;=\; \mathsf{!aCoffee}\,.\,(\mathsf{?coffee} \mid \mathsf{!pay})$$

We have the following type encoding in CCS:

$$[\![T''_{\mathsf{A}}]\!] = \mathsf{!aCoffee}\,.\,\mathsf{!pay}\,.\,\mathsf{?coffee}$$

Then, using the rules in Tables 7.1 and 7.2, we have the following derivation (where $(+\textsc{Ctx})$ *becomes an axiom when $I = \emptyset$ in its premises):*

$$
\dfrac{
\dfrac{
  \dfrac{
    \dfrac{}{\mathbf{0} \mathrel{\ddot{\leqslant}} [\![\mathbf{0}]\!]}\,(+\textsc{Ctx})
  }{
    !\mathsf{pay} \mathrel{\ddot{\leqslant}} [\![!\mathsf{pay}]\!]
  }\,(+\textsc{Ctx})
  \qquad
  \dfrac{
    \dfrac{}{\mathbf{0} \mathrel{\ddot{\leqslant}} [\![\mathbf{0}]\!]}\,(+\textsc{Ctx})
  }{
    ?\mathsf{coffee} \mathrel{\ddot{\leqslant}} [\![?\mathsf{coffee}]\!]
  }\,(+\textsc{Ctx})
  \qquad
  \mathbf{0} \mathrel{\ddot{\approx}} \textstyle\sum_{i \in \emptyset} !\mathsf{c}_i \,.\, Q_i
}{
  ?\mathsf{coffee} \mid !\mathsf{pay} \;\ddot{\approx}\; !\mathsf{pay} \mid ?\mathsf{coffee} \;\ddot{\leqslant}\; [\![!\mathsf{pay}.?\mathsf{coffee}]\!] + \mathbf{0} \;\ddot{\approx}\; [\![!\mathsf{pay}.?\mathsf{coffee}]\!]
}\,(|\mathrm{L})
}{
  ?\mathsf{coffee} \mid !\mathsf{pay} \;\ddot{\leqslant}\; [\![!\mathsf{pay}.?\mathsf{coffee}]\!]
}\,(\ddot{\approx} \times 2)
$$

$$
\dfrac{?\mathsf{coffee} \mid !\mathsf{pay} \;\ddot{\leqslant}\; [\![!\mathsf{pay}.?\mathsf{coffee}]\!]}{!\mathsf{aCoffee} \,.\, (?\mathsf{coffee} \mid !\mathsf{pay}) \;\ddot{\leqslant}\; [\![T''_{\mathsf{A}}]\!]}\,(+\textsc{Ctx})
$$

This shows that we can inductively determine the I/O compliance relation in the conclusion by composing the smaller relations from the premises.

The rules and relations introduced so far do not explicitly mention recursion[3]. In order to syntactically handle recursive terms and asynchrony, however, some additional care is needed. We can notice that rules $(+\textsc{Int})$, $(|\mathrm{R})$ and $(|\mathrm{L})$ allow the removal of branches in output-guarded choices. Under recursion and asynchrony, this can lead to a problem similar to the one illustrated in Example 6.19. Consider, for instance, the following processes[4]:

$$P \;=\; \mu_Y !\mathsf{a} \,.\, Y \qquad Q \;=\; \mu_X !\mathsf{a} \,.\, X + !\mathsf{b} \,.\, ?\mathsf{c} \tag{7.1}$$

If a syntactic rule similar to $(+\textsc{Int})$ is applied to the $+$ sub-term under recursion in Q, such a rule can cause the removal of the $!\mathsf{b}$-guarded branch, thus leading to claim $P[\sigma] \mathrel{\ddot{\leqslant}} Q[\sigma]$; this relation, however, does *not* hold, because $Q[\sigma] \overset{??\mathsf{c}}{\Longrightarrow}$ but $P[\sigma] \overset{?\mathsf{c}}{\not\Longrightarrow}$.

To address this issue without excessively complicating the exposition, we will focus on a fragment of sync/async CCS, introduced in Definition 7.7 below.

Definition 7.7 (CCS$^-$). *CCS$^-$ terms have the following syntax:*

$$P, Q \;::=\; \sum_{i \in I} \ell_{\tau i} \,.\, P_i \;\;\big|\;\; P \mid Q \;\;\big|\;\; X \;\;\big|\;\; \mu_X P$$

where we stipulate that:

a. *if a choice $+$ appears under recursion and some branch contains some input, then* all *recursive branches must contain some input;*

[3]Note that the presence of recursive sub-terms does not influence the validity of Proposition 7.2 and Lemma 7.5, since they are semantically grounded.

[4]Note that P and Q are respectively the encodings of T and U from Example 6.19, by Definition 2.27.

b. *in $\mu_X P$, P is* sequential *(i.e., it cannot contain $|$) and X is* guarded *(i.e., it can only appear within some subterm $\ell \,.\, P'$ of P);*

c. *in $P \mid Q$, either P's or Q's body does not contain inputs.*

We denote the empty summation with $\mathbf{0}$, and will often omit its trailing occurrences. The synchronous and asynchronous semantics of CCS$^-$ are based on the rules shown in Definitions 2.24 and 2.25; they give rise to the behaviours denoted by $\mathbb{U}^-_{CCS}$ and $\mathbb{U}^-_{aCCS}$.

Remark 7.8. *In the rest of this chapter, unless otherwise specified, we will use P, Q, R, S to denote CCS$^-$ terms.*

By Definition 7.7, choices in CCS$^-$ have prefix-guarded branches. Condition *a.* addresses the issue we discussed on page 69, by forbidding processes such as $Q = \mu_X \mathsf{!a} \,.\, X + \mathsf{!b} \,.\, \mathsf{?c}$. Note, however, that processes like $Q' = \mu_X \mathsf{!a} \,.\, \mathbf{0} + \mathsf{!b} \,.\, \mathsf{?c} \,.\, X$ are allowed: in fact, all recursive branches of Q' contain some input, and the only branch without inputs is the *non*-recursive one; as a result, $Q'[\sigma]$ has no persistent inputs that must be preserved on the LHS of I/O simulation. The restriction on $|$ of condition *b.* ensures that the size of CCS$^-$ terms does not grow unbondedly along recursions: hence, each process is finite-state and, in the asynchronous semantics, the only source of infiniteness is the unbounded buffer. This restriction is quite common (see e.g. [Mil89]), and allows for simpler reasoning. Note that e.g. $\mu_X(\mathsf{?a} \mid \mathsf{?b})$ is not valid, but by removing the vacuous recursion operator we have the valid term $\mathsf{?a} \mid \mathsf{?b}$. The guardedness condition for recursion variables comes from our original CCS fragment, and is similar to [Mil89] (§3.2). Finally, condition *c.* enforces the premises of ([|LR]) on all CCS$^-$ processes (albeit not necessary for our main results, this will allow us to aim at Conjecture 10.2 later on, in Section 10.2.2).

Note that, albeit more limited than full CCS, CCS$^-$ is still expressive enough to write our examples from Chapter 3.

The last step towards our inductive system of rules is an induction principle for $\ddot{\lessgtr}$ in $\mathbb{U}^-_{\text{CCS}}$ and $\mathbb{U}^-_{\text{aCCS}}$, formalised in Lemma 7.9 below.

Lemma 7.9. *Let P be a sequential CCS$^-$ term, with X guarded. If $P[{}^{Q}/_{X}] \ddot{\lessgtr} Q$, then $\mu_X P \ddot{\lessgtr} Q$. Moreover, if $\forall\sigma \,.\, P[{}^{Q}/_{X}][\sigma] \ddot{\lessgtr} Q[\sigma]$, then $\forall\sigma \,.\, \mu_X P[\sigma] \ddot{\lessgtr} Q[\sigma]$.*

Proof. See page 140. ☐

Intuitively, Lemma 7.9 captures the behaviour of P w.r.t. Q when reasoning "at the limit" of an infinite sequence of substitutions of X. In CCS$^-$, whenever P is sequential and $R \mathrel{\ddot{\leqslant}} R'$, we have $P[{}^{R}/_{X}] \mathrel{\ddot{\leqslant}} P[{}^{R'}/_{X}]$; therefore, we also have:

$$\begin{aligned}
P^1 = P[{}^{Q}/_{X}] &\mathrel{\ddot{\leqslant}} Q && \text{(by hypothesis)}\\
P^2 = P\left[{}^{P[{}^{Q}/_{X}]}/_{X}\right] &\mathrel{\ddot{\leqslant}} P[{}^{Q}/_{X}] && \text{and thus, } P^2 \mathrel{\ddot{\leqslant}} Q \text{ (by transitivity)}\\
P^3 = P\left[{}^{P\left[{}^{P[{}^{Q}/_{X}]}/_{X}\right]}/_{X}\right] &\mathrel{\ddot{\leqslant}} P\left[{}^{P[{}^{Q}/_{X}]}/_{X}\right] && \text{and thus, } P^3 \mathrel{\ddot{\leqslant}} Q\\
&\vdots && \vdots\\
P^n = P\left[{}^{P\left[{}^{P\left[{}^{\cdots P[{}^{Q}/_{X}]\cdots}/_{X}\right]}/_{X}\right]}/_{X}\right] &\mathrel{\ddot{\leqslant}} P\left[{}^{P\left[{}^{\cdots P[{}^{Q}/_{X}]\cdots}/_{X}\right]}/_{X}\right] && \text{and thus, } P^n \mathrel{\ddot{\leqslant}} Q\\
&\vdots && \vdots
\end{aligned}$$

Informally, in the "limit" behaviour P^∞, the execution of Q is "delayed" by an infinite amount of unfoldings of P: hence, "at the limit", such a behaviour is semantically indistinguishable from (i.e., bisimilar to) $\mu_X P$ — and this, by Theorem 6.11 and transitivity of $\ddot{\leqslant}$, leads us to $\mu_X P \mathrel{\ddot{\leqslant}} Q$.

Definition 7.10. *Let Γ be a mapping from recursion variables to CCS$^-$ terms with all recursion variables being weakly guarded. We define $\ddot{\precsim}_\Gamma$ as the relation between CCS$^-$ terms inductively defined by the rules obtained by replacing $\ddot{\leqslant}$ with $\ddot{\precsim}_\Gamma$ in Table 7.2, and by the following additional rules:*

$$\frac{}{\mathbf{0} \mathrel{\ddot{\precsim}_\Gamma} \mathbf{0}}\ (\text{S-}\mathbf{0}) \qquad \frac{X \notin \operatorname{dom}(\Gamma)}{X \mathrel{\ddot{\precsim}_\Gamma} X}\ (\text{S-}X) \qquad \frac{\Gamma(X) = Q}{X \mathrel{\ddot{\precsim}_\Gamma} Q}\ (\text{S-}\textsc{Var})$$

$$\frac{P \mathrel{\ddot{\precsim}_{\Gamma, X:Q}} Q}{\mu_X P \mathrel{\ddot{\precsim}_\Gamma} Q}\ (\text{S-}\mu\text{L}) \qquad \frac{P \mathrel{\ddot{\precsim}_\Gamma} Q[{}^{\mu_X Q}/_{X}]}{P \mathrel{\ddot{\precsim}_\Gamma} \mu_X Q}\ (\text{S-}\mu\text{R})$$

Moreover, $\ddot{\precsim}_\Gamma$ is closed under replacement of $\equiv$-related subterms (from Definition 7.3). We will often write $P \mathrel{\ddot{\precsim}} Q$ instead of $P \mathrel{\ddot{\precsim}_\emptyset} Q$.

The rules in Definition 7.10 are mostly straightforward:

- (S-$\mathbf{0}$) is the axiom relating terminated processes, which is semantically grounded on the identity relation from Table 7.1. Note that such a rule is not strictly necessary, because it is just a special case of (+Ctx) or (+τ) when $I = \emptyset$: its presence just emphasizes the base case for rule induction;

- (S-X) relates equal open recursion variables, provided that they do *not* appear in the environment. The intuition is that open variables are semantically isomorphic to $\mathbf{0}$;
- (S-Var) uses the environment, requiring the hypothesis that X expands into a behaviour I/O simulated by Q (note that (S-X) and (S-Var) are mutually exclusive);
- (S-μL) consumes such an hypothesis, introducing recursion on the LHS: the intuition is that, in the rule premise, the LHS is I/O simulated by the RHS when Q replaces X in P's body, similarly to the premise of Lemma 7.9;
- (S-μR) allows to unfold a recursion on the RHS, when going upwards in a derivation.

7.2 An I/O simulation-based type system

We can now show that the syntactic rules for $\ddot{\precsim}_\Gamma$ can be a basis for a type system for $\mathbb{U}^-_{\text{aCCS}}$. The correctness of $\ddot{\precsim}$ w.r.t. $\ddot{\leqslant}$ is formalised in Corollary 7.12 below; this, in turn, is based on Theorem 7.11, which shows how the environment in each derivation step is used to construct an intermediate I/O simulation relation.

Theorem 7.11. *Let $P \ddot{\precsim}_\Gamma Q$. Then, $\forall \sigma \;.\; P\Gamma[\sigma] \ddot{\leqslant} Q[\sigma]$.*

Proof. See page 144. □

Corollary 7.12. *Let $P \ddot{\precsim} Q$. Then, $\forall \sigma \;.\; P[\sigma] \ddot{\leqslant} Q[\sigma]$.*

Proof. Follows from Theorem 7.11, when $\Gamma = \emptyset$. □

We can now define a *syntax-directed* typing judgement, relating CCS$^-$ terms with session types. To this purpose, we exploit the encoding in Definition 2.27.

Definition 7.13 (Type system)**.** *We write $\Gamma \vdash P : T$ iff $[\![T]\!]$ is a valid CCS$^-$ term and $P \ddot{\precsim}_\Gamma [\![T]\!]$.*

The condition on $[\![T]\!]$ in Definition 7.13 is necessary because the encoding of recursive session types does not always respect condition *a.* of Definition 7.7: consider, for instance, $U = \text{rec}_Y\, !\mathsf{a}.X \oplus !\mathsf{b}.?\mathsf{c}$ (from Example 6.19) and its encoding $[\![U]\!] = \mu_X !\mathsf{a}\,.\,X + !\mathsf{b}\,.\,?\mathsf{c}$.

Example 7.14. *Recall Alice's type, process and type encoding from Example 7.6, and the $\ddot{\leqslant}$-based derivation therein. By Definition 7.10, we have the following corresponding derivation, where we essentially replace $\ddot{\leqslant}$ with $\ddot{\preccurlyeq}$, $\ddot{\approx}$ with $\equiv$, and the* (+Ctx)*-based axiom with* (S-**0**)*:*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathbf{0}\mathrel{\ddot{\preccurlyeq}}[\![\mathbf{0}]\!]}\,(\text{S-}\mathbf{0})}{\mathsf{!pay}\mathrel{\ddot{\preccurlyeq}}[\![\mathsf{!pay}]\!]}\,(+\textsc{Ctx}) \qquad \dfrac{\dfrac{}{\mathbf{0}\mathrel{\ddot{\preccurlyeq}}[\![\mathbf{0}]\!]}\,(\text{S-}\mathbf{0})}{\mathsf{?coffee}\mathrel{\ddot{\preccurlyeq}}[\![\mathsf{?coffee}]\!]}\,(+\textsc{Ctx}) \qquad \mathbf{0}\equiv\sum_{i\in\emptyset}\mathsf{!c}_i\,.\,Q_i}{\mathsf{?coffee}\mid\mathsf{!pay}\;\equiv\;\mathsf{!pay}\mid\mathsf{?coffee}\mathrel{\ddot{\preccurlyeq}}[\![\mathsf{!pay}.\mathsf{?coffee}]\!]+\mathbf{0}\;\equiv\;[\![\mathsf{!pay}.\mathsf{?coffee}]\!]}\,(|\text{L})}{\mathsf{?coffee}\mid\mathsf{!pay}\mathrel{\ddot{\preccurlyeq}}[\![\mathsf{!pay}.\mathsf{?coffee}]\!]}\,(\equiv\times 2)}{\mathsf{!aCoffee}\,.\,(\mathsf{?coffee}\mid\mathsf{!pay})\mathrel{\ddot{\preccurlyeq}}[\![\mathsf{!aCoffee}.\mathsf{!pay}.\mathsf{?coffee}]\!]}\,(+\textsc{Ctx})}{\vdash P''_{\mathsf{A}}:T''_{\mathsf{A}}}\,(\textsc{Definition}\ 7.13)$$

Example 7.15. *From Chapter 3, recall the bartender's process Q''_{B} that stops selling beer after a certain hour, the bartender type U_{B}, and its encoding in CCS (which is also a valid CCS$^-$ term):*

$$\begin{aligned}
Q''_{\mathsf{B}} \;&=\; \mu_Y\big(\,(\mathsf{?aCoffee}\,.\,\mathsf{!coffee}\,.\,Y \;+\; \mathsf{?aBeer}\;.\;(\mathsf{!beer}\,.\,Y+\mathsf{!no}\,.\,Y)\;+\;\mathsf{?pay})\\
&\qquad\quad +\tau\,.\,\mu_Z(\mathsf{?aCoffee}\,.\,\mathsf{!coffee}\,.\,Z\;+\;\mathsf{?aBeer}\,.\,\mathsf{!no}\,.\,Z\;+\;\mathsf{?pay})\,\big)\\
U_{\mathsf{B}} \;&=\; \mathrm{rec}_X\,(\mathsf{?aCoffee}.\mathsf{!coffee}.X\;\;\&\;\;\mathsf{?aBeer}.(\mathsf{!beer}.X\oplus\mathsf{!no}.X)\;\;\&\;\;\mathsf{?pay})\\
[\![U_{\mathsf{B}}]\!] \;&=\; \mu_X(\mathsf{?aCoffee}\,.\,\mathsf{!coffee}\,.\,X\;+\;\mathsf{?aBeer}\;.\;(\mathsf{!beer}\,.\,X+\mathsf{!no}\,.\,X)\;+\;\mathsf{?pay})
\end{aligned}$$

We want to prove $\vdash Q''_{\mathsf{B}}:U_{\mathsf{B}}$. First of all, for Q''_{B} we show a typing derivation for the term under Z-recursion $\mu_Z\mathsf{?aCoffee}\ldots$. Letting $\Gamma = Z:[\![U_{\mathsf{B}}]\!]$, we have the derivation shown in Table 7.3. Secondly, let $\mathcal{D}$ indicate such a derivation, starting from the instance of rule (S-μL) *(i.e., excluding the application of Definition 7.13). We reuse $\mathcal{D}$ in the derivation for the term under Y-recursion in Q''_{B}, where it provides a premise for applying rule* (+Ext) *from Table 7.2. Letting $\Gamma' = Y:[\![U_{\mathsf{B}}]\!]$, we obtain the derivation in Table 7.4*[5].

Theorem 7.16 below states the correctness of our typing discipline. Suppose you have a process P with type T, and a process Q with type U. If $T[]$ and $U[]$ are I/O compliant, then $P[]$ and $Q[]$ are I/O compliant, too. Thus, we have that $P[]\parallel Q[]$ is safe (by Theorem 5.22).

Theorem 7.16 (Correctness)**.** *If $\vdash P:T$ and $\vdash Q:U$ with $T[]\mathrel{\ddot{\bowtie}}U[]$, then $P[]\mathrel{\ddot{\bowtie}}Q[]$.*

[5]In this derivation, the environment Γ' is larger than the empty environment at the root of $\mathcal{D}$. However, $\mathcal{D}$ "still works", and can be simply rewritten with Γ' at its root, by Proposition D.4.

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{\Gamma(Z) = [\![U_{\mathsf{B}}]\!]}{Z \;\ddot{\precsim}_\Gamma\; [\![U_{\mathsf{B}}]\!]}\;(\textsc{S-Var})
}{\mathsf{!coffee}\,.\,Z \;\ddot{\precsim}_\Gamma\; [\![\mathsf{!coffee}\,.\,U_{\mathsf{B}}]\!]}\;(+\textsc{Ctx})
}{\mathsf{?aCoffee}\,.\,\mathsf{!coffee}\,.\,Z \;\ddot{\precsim}_\Gamma\; [\![\mathsf{?aCoffee}.\mathsf{!coffee}\,.\,U_{\mathsf{B}}]\!]}\;(+\textsc{Ctx})
\quad
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{\Gamma(Z) = [\![U_{\mathsf{B}}]\!]}{Z \;\ddot{\precsim}_\Gamma\; [\![U_{\mathsf{B}}]\!]}\;(\textsc{S-Var})
}{\mathsf{!no}\,.\,Z \;\ddot{\precsim}_\Gamma\; [\![\mathsf{!no}.U_{\mathsf{B}}]\!]}\;(+\textsc{Ctx})
}{\mathsf{!no}\,.\,Z \;\ddot{\precsim}_\Gamma\; [\![\mathsf{!beer}.U_{\mathsf{B}} \oplus \mathsf{!no}.U_{\mathsf{B}}]\!]}\;(+\textsc{Int})
}{\mathsf{?aBeer}\,.\,\mathsf{!no}\,.\,Z \;\ddot{\precsim}_\Gamma\; [\![\mathsf{?aBeer}.(\mathsf{!beer}.U_{\mathsf{B}} \oplus \mathsf{!no}.U_{\mathsf{B}})]\!]}\;(+\textsc{Ctx})
\quad
\dfrac{
\dfrac{}{\mathbf{0} \;\ddot{\precsim}_\Gamma\; [\![\mathbf{0}]\!]}\;(\textsc{S-}\mathbf{0})
}{\mathsf{?pay} \;\ddot{\precsim}_\Gamma\; [\![\mathsf{?pay}]\!]}\;(+\textsc{Ctx})
}{\mathsf{?aBeer}\,.\,\mathsf{!no}\,.\,Z + \mathsf{?pay} \;\ddot{\precsim}_\Gamma\; [\![\mathsf{?aBeer}.(\mathsf{!beer}.U_{\mathsf{B}} \oplus \mathsf{!no}.U_{\mathsf{B}})\ \&\ \mathsf{?pay}]\!]}\;(+\textsc{Ctx})
}{\mathsf{?aCoffee}\,.\,\mathsf{!coffee}\,.\,Z + \mathsf{?aBeer}\,.\,\mathsf{!no}\,.\,Z + \mathsf{?pay} \quad \ddot{\precsim}_\Gamma \quad [\![\mathsf{?aCoffee}.\mathsf{!coffee}\,.\,U_{\mathsf{B}}\ \&\ \mathsf{?aBeer}.(\mathsf{!beer}.U_{\mathsf{B}} \oplus \mathsf{!no}.U_{\mathsf{B}})\ \&\ \mathsf{?pay}]\!]}\;(+\textsc{Ext})
$$

$$
\dfrac{
\dfrac{
\dfrac{\cdots}{\mathsf{?aCoffee}\,.\,\mathsf{!coffee}\,.\,Z + \mathsf{?aBeer}\,.\,\mathsf{!no}\,.\,Z + \mathsf{?pay} \;\ddot{\precsim}_\Gamma\; [\![U_{\mathsf{B}}]\!]}\;(\textsc{S-}\mu\textsc{R})
}{\mu_Y\mathsf{?aCoffee}\,.\,\mathsf{!coffee}\,.\,Z + \mathsf{?aBeer}\,.\,\mathsf{!no}\,.\,Z + \mathsf{?pay} \;\ddot{\precsim}\; [\![U_{\mathsf{B}}]\!]}\;(\textsc{S-}\mu\textsc{L})
}{\vdash \mu_Z\mathsf{?aCoffee}\,.\,\mathsf{!coffee}\,.\,Z + \mathsf{?aBeer}\,.\,\mathsf{!no}\,.\,Z + \mathsf{?pay} : U_{\mathsf{B}}}\;(\textsc{Definition}\ 7.13)
$$

Table 7.3: Typing derivation for the running example (I).

.

$$
\dfrac{
\dfrac{
\dfrac{\dfrac{\Gamma'(Y) = [\![U_{\mathsf{B}}]\!]}{Y \mathrel{\ddot{\preccurlyeq}_{\Gamma'}} [\![U_{\mathsf{B}}]\!]}\;(\text{S-}\textsc{Var})}{\mathsf{!coffee} \,.\, Y \mathrel{\ddot{\preccurlyeq}_{\Gamma'}} [\![\mathsf{!coffee} \,.\, U_{\mathsf{B}}]\!]}\;(+\textsc{Ctx})
\quad
\dfrac{\dfrac{
\dfrac{\dfrac{\Gamma'(Y) = [\![U_{\mathsf{B}}]\!]}{Y \mathrel{\ddot{\preccurlyeq}_{\Gamma'}} [\![U_{\mathsf{B}}]\!]}\;(\text{S-}\textsc{Var})}{\mathsf{!beer} \,.\, Y \mathrel{\ddot{\preccurlyeq}_{\Gamma'}} [\![\mathsf{!beer}.U_{\mathsf{B}}]\!]}\;(+\textsc{Ctx})
\quad
\dfrac{\dfrac{\Gamma'(Y) = [\![U_{\mathsf{B}}]\!]}{Y \mathrel{\ddot{\preccurlyeq}_{\Gamma'}} [\![U_{\mathsf{B}}]\!]}\;(\text{S-}\textsc{Var})}{\mathsf{!no} \,.\, Y \mathrel{\ddot{\preccurlyeq}_{\Gamma'}} [\![\mathsf{!no}.U_{\mathsf{B}}]\!]}\;(+\textsc{Ctx})
}{\mathsf{!beer} \,.\, Y + \mathsf{!no} \,.\, Y \mathrel{\ddot{\preccurlyeq}_{\Gamma'}} [\![\mathsf{!beer}.U_{\mathsf{B}} \oplus \mathsf{!no}.U_{\mathsf{B}}]\!]}\;(+\textsc{Ctx})
}{\mathsf{?aBeer} \,.\, (\mathsf{!beer} \,.\, Y + \mathsf{!no} \,.\, Y) \mathrel{\ddot{\preccurlyeq}_{\Gamma'}} [\![\mathsf{?aBeer}.(\mathsf{!beer}.U_{\mathsf{B}} \oplus \mathsf{!no}.U_{\mathsf{B}})]\!]}\;(+\textsc{Ctx})
\quad
\dfrac{\dfrac{}{\mathbf{0} \mathrel{\ddot{\preccurlyeq}_{\Gamma'}} [\![\mathbf{0}]\!]}\;(\text{S-}\mathbf{0})}{\mathsf{?pay} \mathrel{\ddot{\preccurlyeq}_{\Gamma'}} [\![\mathsf{?pay}]\!]}\;(+\textsc{Ctx})
\quad
\mathcal{D}
}{\mathsf{?aCoffee} \,.\, \mathsf{!coffee} \,.\, Y + \mathsf{?aBeer} \,.\, (\mathsf{!beer} \,.\, Y + \mathsf{!no} \,.\, Y) + \mathsf{?pay} + \tau \,.\, \mu_Z \mathsf{?aCoffee} \,.\, \mathsf{!coffee} \,.\, Z + \mathsf{?aBeer} \,.\, \mathsf{!no} \,.\, Z + \mathsf{?pay} \mathrel{\ddot{\preccurlyeq}_{\Gamma'}} [\![U_{\mathsf{B}}]\!]}\;(+\textsc{Ext},\ \text{S-}\mu\text{R})
}{
\dfrac{Q''_{\mathsf{B}} \mathrel{\ddot{\preccurlyeq}} [\![U_{\mathsf{B}}]\!]}{\vdash Q''_{\mathsf{B}} : U_{\mathsf{B}}}\;(\textsc{Definition } 7.13)
}\;(\text{S-}\mu\text{L})
$$

Table 7.4: Typing derivation for the running example (II). $\mathcal{D}$ is the derivation starting from (S-μL) in Table 7.3.

.

Proof. From Definition 7.13 we have $P \mathrel{\ddot{\precsim}} [\![T]\!]$; by Proposition 2.28, Definition 7.10 and Corollary 7.12 it follows $P[] \mathrel{\ddot{\leqslant}} T[]$. Similarly, $Q[] \mathrel{\ddot{\leqslant}} U[]$. Since $P[] \mathrel{\ddot{\leqslant}} T[] \mathrel{\ddot{\bowtie}} U[]$, by Theorem 6.13 it follows $P[] \mathrel{\ddot{\bowtie}} U[]$. Since $Q[] \mathrel{\ddot{\leqslant}} U[] \mathrel{\ddot{\bowtie}} P[]$, then by Theorem 6.13 we conclude $Q[] \mathrel{\ddot{\bowtie}} P[]$. □

We stress that the above result is obtained just by exploiting the properties of I/O simulation, without explicitly proving subject reduction.

In the synchronous setting, we can deduce $T \mathrel{\ddot{\bowtie}} U$ either via model checking (since both behaviours are finite state), or using syntax-driven techniques (e.g. those in [BL10]); then, by Proposition 4.12, this result is lifted "for free" to the async case; and if both T and U can be encoded in CCS$^-$, we can reason on $\vdash P : T$ and $\vdash Q : U$ on a syntax-driven basis, through the rules in Definition 7.10.

Note, however, that Theorem 7.16 does *not* require compliance between *synchronous* session types. Therefore, the result also holds e.g. for $T = !\mathsf{a}.?\mathsf{b}$ and $U = !\mathsf{b}.?\mathsf{a}$ — since in the asynchronous setting we have $T[] \mathrel{\ddot{\bowtie}} U[]$ (even though $T \not\mathrel{\ddot{\bowtie}} U$). This aspect is further discussed in Examples 7.17 and 7.18 below.

Example 7.17. *Recall* $\vdash P''_{\mathsf{A}} : T''_{\mathsf{A}}$ *from Example 7.14, and consider the bartender processes* $Q_{\mathsf{B}}, Q''_{\mathsf{B}}$ *and type* U_{B} *from Chapter 3. We can easily obtain* $\vdash Q_{\mathsf{B}} : U_{\mathsf{B}}$*. Therefore, since in Example 4.10 we determined* $U_{\mathsf{B}}[] \mathrel{\ddot{\bowtie}} T''_{\mathsf{A}}[]$*, by Theorem 7.16 we have* $Q_{\mathsf{B}}[] \mathrel{\ddot{\bowtie}} P''_{\mathsf{A}}[]$*; hence, by Theorem 5.22 we have that* $Q_{\mathsf{B}}[] \parallel P''_{\mathsf{A}}[]$ *is safe. Note that this result exploits asynchrony* both *via I/O compliance and via typing: in fact, this is based on* $U_{\mathsf{B}}[] \mathrel{\ddot{\bowtie}} T''_{\mathsf{A}}[]$*, albeit* $U_{\mathsf{B}} \not\mathrel{\ddot{\bowtie}} T''_{\mathsf{A}}$ *(as discussed in Example 4.10); moreover, the typing judgement* $\vdash P''_{\mathsf{A}} : T''_{\mathsf{A}}$ *uses rule* $(\lfloor\mathrm{L})$.

Example 7.18. *Recall* $\vdash P''_{\mathsf{A}} : T''_{\mathsf{A}}$ *from Example 7.14, and consider Example 7.15, where we show* $\vdash Q''_{\mathsf{B}} : U_{\mathsf{B}}$*. By Theorem 7.16 we have* $Q''_{\mathsf{B}}[] \mathrel{\ddot{\bowtie}} P''_{\mathsf{A}}[]$*, and by Theorem 5.22 we conclude that* $Q''_{\mathsf{B}}[] \parallel P''_{\mathsf{A}}[]$ *is safe. Note that this result, as in Example 7.17, is based on* asynchronous *semantics, and would not hold in the synchronous setting.*

The previous examples (in particular, Example 7.15) show that our syntax-driven rules allow to type an Erlang-style `receive`...`after`... behaviour, featured in the bartender process.

Chapter 8

The LTS Workbench

`LTSwb` (from *"`LTS WorkBench`"*) [Sca15; SB15] is a Labelled Transition System (LTS) toolbox, allowing to define LTSs and processes, and compute relations between their states. It has been mainly implemented to experiment with the theory presented in this work, and to ease further investigations on semantic and language-independent relations. Its main features are:

genericity. `LTSwb` does not require LTSs and processes to have specific state/label types. This allows to semantically reason on different process specifications: for example, it allows to study whether a CCS process [Mil89] is a semantic refinement of a session type [Hon93] (as in the present work), or whether it can correctly interact with a service whose specification is a Communicating Finite-State Machine (CFSM) [BZ83];

laziness. Finite-state and infinite-state LTSs and processes are managed transparently: states and transitions are only generated upon request. This allows to handle the state space explosion problems and the infiniteness arising e.g. with recursion, parallelism, unbounded communication buffers, *etc.*

`LTSwb` is a Scala [Oa04] library. The choice of Scala comes from the desire of a functional programming language with an advanced type system, and the possibility of accessing the vast landscape of libraries available on the Java VM. `LTSwb` can also be used on the interactive Scala console: unless otherwise noted, all the examples on this chapter can be replicated therein via simple cut&pasting.

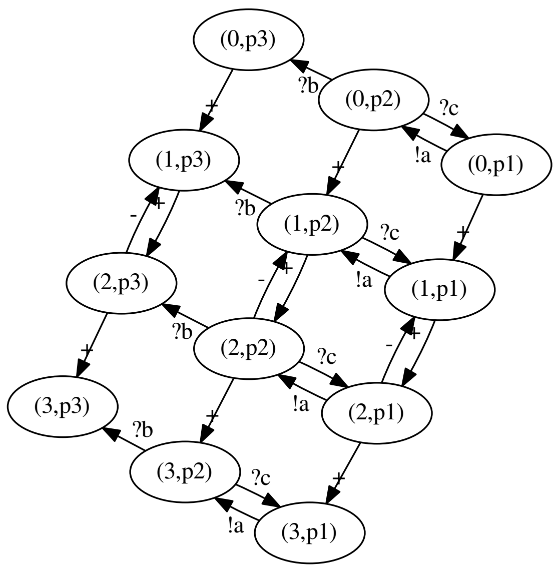


Figure 8.1: Output of `(l1 ||| l2).toDot`.

8.1 LTSs, processes and asynchrony

In `LTSwb`, an *LTS* is a triple $(\Sigma, \Lambda, \mathcal{R})$ where Σ is the set of *states*, Λ is the set of *labels*, and $\mathcal{R} \subseteq (\Sigma \times (\Lambda \times \Sigma))$ is the *transition relation*. A *process* is a pair (L, σ) where L is an LTS and σ is one of its states. The *process transition* $(L, \sigma) \xrightarrow{\ell} (L, \sigma')$ holds iff $(\sigma, (\ell, \sigma'))$ is in the transition relation of L.

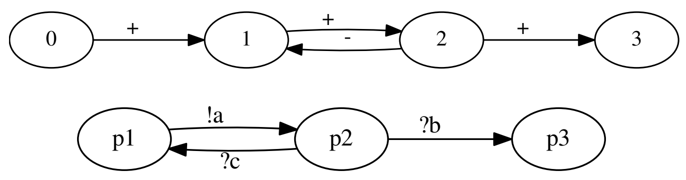
In the following sections, we show several ways in which `LTSwb` processes can be created (by extracting them from some LTS) and manipulated.

8.1.1 From LTSs to processes

In `LTSwb`, a finite LTS can be defined with the `LTS` constructor, by enumerating the state-(label-state) triples which compose its transition relation. For example:

```
val l1 = LTS(List((0, ("+", 1)), (1, ("+", 2)), (2, ("+", 3)), (2, ("-", 1))))
val l2 = LTS(List(("p1", ("!a", "p2")), ("p2", ("?b", "p3")), ("p2", ("?c", "p1"))))
```

The type of `l1` is `FiniteLTS[Int,String]`, while `l2` has type and `FiniteLTS[String,String]` — i.e., they are finite-state, finite-branching LTSs where states are `Integers` (resp. `Strings`),



and labels are `Strings`. The methods `l1.toDot` and `l2.toDot` return their graphs (shown on the left). The `|||` operator on LTSs returns the LTS whose states correspond to the parallel composition of its arguments' states, provided that the labels have the same type: Figure 8.1 shows the diagram of `(l1 ||| l2).toDot`. Such a composition corresponds to Definition 2.4, and it is performed *lazily*, thus avoiding (or delaying) state space explosion problems: the actual combinations of LTS states are generated only upon request.

A process can be simply retrieved from an LTS through one of its states. For example:

```
val p1 = l2.process("p1")
```

In this case, we have that `p1` has type `FiniteProcess[String,String]` (i.e., a finite-state, finite-branching process where states are `String`s, and labels are `String`s as well). As one might expect, `p1.state` has indeed value `"p1"`. Moreover, `p1.lts` is `l2` — i.e., the LTS inhabited by `p1`.

A process can be queried for its enabled transitions. In our example, `p1.transitions` has type `FiniteSet[String]`, and value `Set("!a")`. We can now let:

```
val p1a = p1("!a");   val p2 = p1a.iterator.next
```

where `p1a` is the `FiniteSet` of processes reachable from `p1` via transition `"!a"`. In our example, `p1a` contains a single element, i.e. the process corresponding to state `"p2"` of `l2`: such a process is retrieved via `p1a`'s iterator[1], and assigned to `p2`. As expected, `p2.transitions` has value `Set("?b","?c")`.

Processes can be composed in parallel, similarly to LTSs (as shown above). Let:

```
val p01 = l1.process(0) ||| p1
```

Here, `p10` has type `FiniteProcess[(Int,String),String]` (i.e., each state is a *pair* of type `(Int,String)`, while labels remain `String`s). The transitions of `p01` are those of the LTS state `(0,p1)` in Figure 8.1: indeed, the same process could have been extracted from the `(l1|||l2)` LTS with `(l1|||l2).process((0,"p1"))`, and `p01.lts` is `l1|||l2`.

8.1.2 CCS processes

`LTSwb` implements `CCS`, which is the *infinite* LTS where states are `CCSTerm`s, labels are `CCSPrefix`es, and the (infinite) transition relation corresponds to the CCS semantics.

[1]Note that the same process can also be retrieved via `l2.process("p2")`, as we did for `p1` above.

Processes can be extracted from `CCS` as above, i.e. with `CCS.process(s)` (where `s` is a `CCSTerm`), or letting `LTSwb` parse terms from strings:

```
val ccs1 = CCS.process("rec(X)(!a.(?b + ?c.X))") // Parses the CCSTerm from String
val ccs2 = CCS("?a.(t.!c.?a.!b + t.!b)") // Shorthand. "t" is the internal action
```

The type of `ccs1` and `ccs2` is `FiniteBranchingProcess[CCSTerm,CCSPrefix]` — i.e., they are finite-branching (but *not* necessarily finite-state) processes whose states are `CCSTerm`s, and whose transition labels are `CCSPrefix`es. Note that `ccs1` has, intuitively, the same transitions of process `p1` defined earlier: for example, `ccs1.transitions` is `Set(!a)`. There is, however, a difference: `CCSPrefix`es are distinguished among *input*, *output* and *internal* actions (respectively: $?\mathtt{a}, !\mathtt{a}, \tau$ — just as in our I/O LTS from Section 2.1.1), and this additional information (which is *not* present in the simple string labels of `p1` above) allows the parallel composition of CCS processes to synchronise. For example, let:

```
val ccs12 = ccs1 ||| ccs2
```

Here, `ccs12` has type `FiniteBranchingProcess[(CCSTerm,CCSTerm),CCSPrefix]`, and the value of `ccs12.transitions` is `Set(?a, !a, τ)`. As expected, the τ-transition is generated by the synchronisation on `a` — and indeed, as shown in Figure 8.3, `ccs12(τ)` returns[2]:

```
Set( ( (?b + (?c.rec(X)(!a.(?b + ?c.X))) , (t.!c.?a.!b + t.!b) ) )
```

8.1.3 From synchronous to asynchronous semantics

If `p` is an instance of `Process` (which is the main abstract class common to *all* `LTSwb` processes), then `p.async` is a new process obtained by pairing `p` with an empty *buffer*, represented as a `List`. `LTSwb` performs this transformation in a general, purely semantic fashion[3]: each *output* label of `p` is appended to the buffer (with an internal transition), and the *head* of the buffer enables a corresponding output transition. This change is transparently reflected in the values returned by `p.async.transitions`. If `p` has been created with the `CCS(`P`)` constructor (where P is a CCS term), then the semantics of `p.async` corresponds to $P[]$, as per Definition 2.25 — although there is no async-CCS-specific code for this functionality. For example:

```
val ccs1a = ccs1.async;   val ccs2a = ccs2.async
```

[2]Note that `ccs12(τ)` and its return value have been slightly edited for clarity, and thus are *not* valid Scala code.

[3]Indeed, such an operation is performed at the LTS level: if `l` is an `LTS`, then `l.async` is the `LTS` with `l`'s states paired with a buffer; if `s` is a state of `l`, then `l.async.process((s, List()))` is equal to `l.process(s).async`.

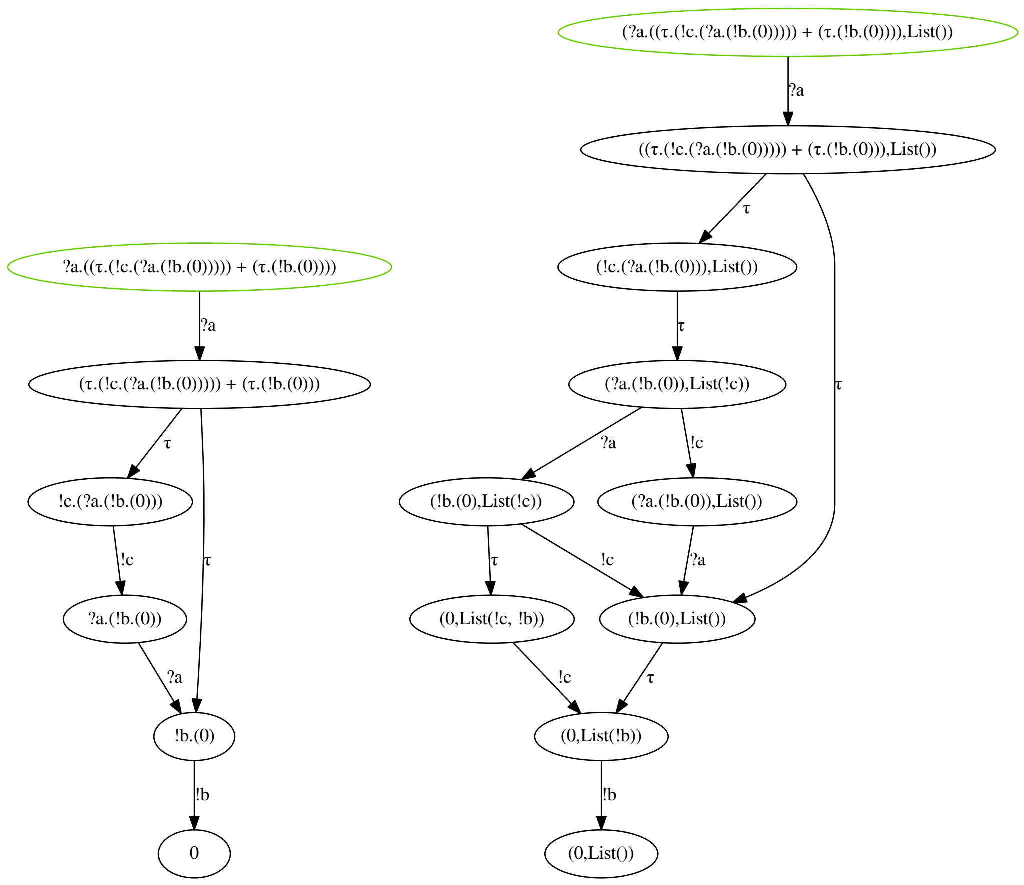

?a.((τ.(!c.(?a.(!b.(0))))) + (τ.(!b.(0))))
?a
(τ.(!c.(?a.(!b.(0))))) + (τ.(!b.(0)))
τ
!c.(?a.(!b.(0)))
!c
?a.(!b.(0))
τ
?a
!b.(0)
!b
0
(?a.((τ.(!c.(?a.(!b.(0))))) + (τ.(!b.(0)))),List())
?a
((τ.(!c.(?a.(!b.(0))))) + (τ.(!b.(0))),List())
τ
(!c.(?a.(!b.(0))),List())
τ
τ
(?a.(!b.(0)),List(!c))
?a
!c
(!b.(0),List(!c))
(?a.(!b.(0)),List())
τ
!c
?a
(0,List(!c, !b))
(!b.(0),List())
!c
τ
(0,List(!b))
!b
(0,List())

Figure 8.2: Outputs of `ccs2.toDot()` (left) and `ccs2.toDot()` (right).

`ccs1a` and `ccs2a` have type `FiniteBranchingProcess[(CCSTerm,Seq[CCSPrefix]),CCSPrefix]` (i.e., each state pairs a `CCSTerm` with a sequence of prefixes). The difference between `ccs2` and `ccs2a` is shown in Figure 8.2: it can be seen that, for example, the first `!c` transition of `ccs2` becomes a τ transition (with buffering) in `ccs2a`, and the head of the buffer is later consumed with a `!c` transition. Note, however, that there is an important difference between `ccs1` and `ccs1a`: while the former has a *finite* number of states, the latter has *infinite* states, due to the presence of recursion and unbounded buffers (the difference can be seen in Figure 8.4). This is not a problem *per se*, because, as shown above, `LTSwb` ensures that process transitions are expanded "lazily". Pairing a finite processes with an unbounded buffer reminds of Communicating Finite State Machines (CFSMs) [BZ83] — and indeed, a CFSM-like interaction (modulo the different naming of labels) can be modeled with the composition `ccs1a ||| ccs2a`, by filtering the states reachable via internal moves and synchronisations: the resulting finite transition diagram is shown in Figure 8.5 (note that the "unfiltered" transition diagram of `ccs1a ||| ccs2a` is infinite).

```
val alice = CCS("!aCoffee.?coffee.!pay + !aBeer.(?beer.!pay + ?no.!pay)")
val bartender = CCS("rec(Y)(?aCoffee.!coffee.Y + ?aBeer.(!beer.Y + !no.Y) + ?pay)")
val ab = IOCompliance.build(alice, bartender)
val aba = IOCompliance.build(alice.async, bartender.async)
```

Listing 8.1: `LTSwb` example. Alice and bartender CCS processes are from Chapter 3.

8.1.4 Adding new process calculi

`LTSwb` has no "hardwired" notion of process calculus. A new process calculus with labelled semantics can be added to the framework in three steps: (a) define (or possibly reuse) a class `L` for its labels, (b) define a class `T` for its terms, and (c) suitably derive the abstract classes `LTS` and `Process`, using `T` and `L` respectively as state and label types (eventually specifying which labels are input/output/internal, and how they synchronise). This very approach has been followed for implementing `CCS` under `LTSwb`[4]: as a result, the CCS-specific code is mostly necessary for parsing terms, while all the operations on CCS processes (e.g.`|||`, `.toDot()`, `.async`,...) are implemented generically — and thus, are reusable for new calculi. Moreover, if two processes (notwithstanding their LTS) share the same label type, then they can synchronise, and their relations can be studied as shown in Section 8.2.

8.2 Behavioural relations

One of the goals of `LTSwb` is implementing and studying *semantic* relations, without syntactic limitations. `LTSwb` currently implements (bi)simulation (Definition 2.11), and several variants of *progress* (Definition 4.1) and *I/O compliance* (Definition 4.4), i.e. notions of "correct" interaction between processes. We exemplify the latter (the others are used similarly).

8.2.1 Experiments with I/O compliance

The `IOCompliance.build()` method takes two `FiniteBranchingProcess` instances p and q, and returns an `Either` object whose `Right` value is a *finite* I/O compliance relation (as per Definition 4.4.) If p, q are *not* I/O compliant, the returned `Left` value is a *counterexample*, i.e. a pair of non-I/O compliant states. Consider the first call to

[4]With an additional trick: `CCSTerm`s are *also* `Process`es, such that if `t` is a `CCSTerm`, then `t.lts` is `CCS`.

```
val aliceH = CCS("!aCoffee.(?coffee | !pay)")
val bartenderL = CCS("rec(Y)(?aCoffee.!coffee.Y + ?aBeer.(!beer.Y + !no.Y) + ?pay
                             + t . rec(Z)(?aCoffee.!coffee.Z + ?aBeer.!no.Z + ?pay))")
val aHbL = IOCompliance.build(aliceH, bartenderL)
val aHbLa = IOCompliance.build(aliceH.async, bartenderL.async)
```

Listing 8.2: Another `LTSwb` example: Alice tries to grab the coffee and pay at the same time; the bartender, instead, may stop selling beer (from Chapter 3).

`IOCompliance.build()` in Listing 8.1: since `alice` and `bartender` are I/O compliant, `ab`'s `Right` value is an I/O compliance relation containing the pair (`alice`, `bartender`); the same holds for `aba`, built on the *asynchronous* versions of the two processes.

Listing 8.2 shows more examples: `aliceH` corresponds to P''_{A} in Chapter 3, while `bartenderL` corresponds to Q''_{B}. The *second* call to `IOCompliance.build()` is successful and returns `Right`, with an I/O compliance relation containing the *asynchronous* processes. This result is coherent with Example 7.18. The *first* call to `IOCompliance.build()`, instead, is *not* successful, and `aHbL` is the `Left` value below (edited for clarity):

```
Left( (?coffee | !pay ),
      (!coffee.rec(Y)(?aCoffee.!coffee.Y + ?aBeer.(!beer.Y + !no.Y) + ?pay
                              + t.rec(Z)(?aCoffee.!coffee.Z + ?aBeer.!no.Z + ?pay))) )
```

The problem is that, after synchronising on $\mathsf{aCoffee}$, `aliceH` and `bartenderL` reach the states inside `Left(···)`, where the $!\mathsf{pay}$ transition of the former is *not* matched by a (weak) $?\mathsf{pay}$ of the latter. This violates clause *a.* of Definition 4.4.

8.2.2 Adding new compliance relations

Both `IOCompliance` and `Progress` (and their asymmetric versions) are derivatives of an abstract, reusable class called `Compliance`. Intuitively, $\mathcal{R}$ is a coinductive *compliance relation* iff, whenever $(p, q) \in \mathcal{R}$, then:

1. `pred(`p`,`q`)` holds; (where `pred` is given as a parameter)

2. $p \xrightarrow{\ell} p'$ and $q \xrightarrow{\ell'} q'$ and ℓ, ℓ' can synchronise implies $(p', q') \in \mathcal{R}$;

3. $p \Rightarrow p'$ and $q \Rightarrow q'$ implies $(p', q') \in \mathcal{R}$. (where $\Rightarrow$ represents 0 or more internal moves)

`Compliance` implements the `.build()` method according to the definition above: given (p, q), it ensures that a class-specific predicate `pred` holds for p, q (as per clause *1.*), and then checks their reducts after synchronisation or internal moves (as per clauses *2.* and

3.). `Compliance.build()` terminates when either no more states need to be checked, or `pred` is false: in the latter case, it returns a counterexample, as seen in Section 8.2.1. For example, the `IOCompliance`-specific predicate matches clause *a.* of Definition 4.4 (in the symmetric variant), and `.build()` ensures that it holds for each pair of states in the relation. `Progress`, `IOCompliance` and their variants are implemented by just changing `pred`, and new coinductive compliance relations can be added in the same way: e.g., the *"Correct contract composition"* from [BZ07a] (Def. 3) can be added by defining `pred(`p`,`q`)` as `(`p `|||` q`).wbarbs.contains(✓)` (where `.wbarbs` is the `Set` of weak barbs of a process, and ✓ is a label denoting success).

Note that `Compliance.build()` only implements a *semi*-algorithm: hence, the method *may* not terminate if one of the processes under analysis is infinite-state — and in particular, if it can reduce, through internal moves, to an infinite number of distinct states. In such a situation, `LTSwb` may need to construct an *infinite* compliance relation, with an infinite search for states violating `pred`. Our Alice/bartender examples are infinite-state, but do not generate infinite internal moves, and the semi-algorithm terminates.

Verifying relations. `LTSwb` also implements the method `Compliance.check()`. Given an instance `r` of some `Compliance`-derived relation, `r.check()` is `true` when each pair of states in `r` actually respects `pred` according to clause *1.* above, and `r` contains all the pairs of states required by clauses *2.* and *3.*. Consider e.g. Listing 8.1: `ab` is a `Right` value, and `ab.right.get.check()` is `true`, because for each pair of states, `pred` (i.e., clause *a.* of Definition 4.4) is satisfied, and the same holds for their τ-reducts according to clauses *2.* and *3.* (which correspond to clauses *b.*–*d.* of Definition 4.4). This also holds for `aba`, and `aHbLa` from Listing 8.2. It is important to note that `Compliance.build()` and `Compliance.check()` are implemented *separately*: the latter is intended as an independent verification method, also for relations which are defined "by hand" (i.e., directly as finite sets of pairs of states) *without* resorting to their own `.build()` method[5]. For example, we can instantiate a `Progress` relation from an existing relation:

```
val aHbLaProg = Progress(aHbLa.right.get) // Recall: aHbLa is an IOCompliance rel.
```

and in this case `aHbLaProg.check()` holds — i.e., notwithstanding its type, `aHbLa` is *also* a progress relation (as expected by item (a) of Theorem 4.9). Under this framework,

[5]When debugging is enabled, `LTSwb` runs `.check()` on *each* relation created by `Compliance.build()`, to test its code.

if a new compliance relation is implemented as explained above (i.e., by deriving the `Compliance` class and providing a suitable class-specific `pred`), then synthesis (`.build()`) and verification (`.check()`) are obtained "for free". A similar framework is also in place for (bi)simulation.

8.3 Conclusions and future work on `LTSwb`

In the current (early) stage of development, `LTSwb` offers a flexible and extensible platform allowing to define generic LTSs and processes, explore their (finite or infinite) state space and study their (bi)simulation and compliance relations. The most similar tool, albeit more CCS-centric, is [CPS93], whose development stopped around 1999: hence, its obsolete dependencies and restrictive licensing terms make it very difficult to use and improve. Another related tool is *LTS Analyser* [MK06] — which is limited to finite-state processes; moreover, its development stopped around 2006, and its source code is not available.

Future work on `LTSwb` includes the addition of more relations, with a "reusable" approach to synthesis and verification similar to the one adopted for `Compliance` and (bi)simulation. Moreover, we plan better support for multiparty interactions (currently provided via the `PCCS` calculus, not discussed here) and richer process calculi with time and value passing. We also plan to integrate `LTSwb` with Gephi [Gep15], thus providing a better user interface with interactive exploration of large transition diagrams.

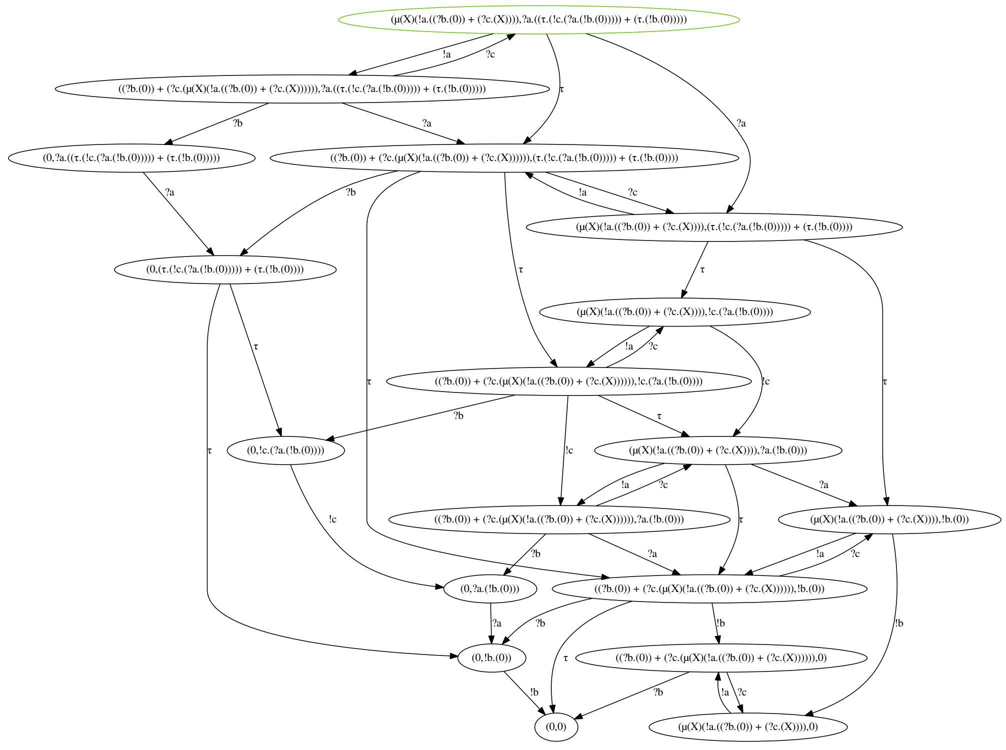

(μ(X)(!a.((?b.(0)) + (?c.(X)))),?a.((τ.(!c.(?a.(!b.(0))))) + (τ.(!b.(0)))))
((?b.(0)) + (?c.(μ(X)(!a.((?b.(0)) + (?c.(X)))))),?a.((τ.(!c.(?a.(!b.(0))))) + (τ.(!b.(0)))))
(0,?a.((τ.(!c.(?a.(!b.(0))))) + (τ.(!b.(0)))))
((?b.(0)) + (?c.(μ(X)(!a.((?b.(0)) + (?c.(X)))))),(τ.(!c.(?a.(!b.(0))))) + (τ.(!b.(0))))
(μ(X)(!a.((?b.(0)) + (?c.(X)))),(τ.(!c.(?a.(!b.(0))))) + (τ.(!b.(0))))
(0,(τ.(!c.(?a.(!b.(0))))) + (τ.(!b.(0))))
(μ(X)(!a.((?b.(0)) + (?c.(X)))),!c.(?a.(!b.(0))))
((?b.(0)) + (?c.(μ(X)(!a.((?b.(0)) + (?c.(X)))))),!c.(?a.(!b.(0))))
(0,!c.(?a.(!b.(0))))
(μ(X)(!a.((?b.(0)) + (?c.(X)))),?a.(!b.(0)))
((?b.(0)) + (?c.(μ(X)(!a.((?b.(0)) + (?c.(X)))))),?a.(!b.(0)))
(μ(X)(!a.((?b.(0)) + (?c.(X)))),!b.(0))
(0,?a.(!b.(0)))
((?b.(0)) + (?c.(μ(X)(!a.((?b.(0)) + (?c.(X)))))),!b.(0))
(0,!b.(0))
((?b.(0)) + (?c.(μ(X)(!a.((?b.(0)) + (?c.(X)))))),0)
(0,0)
(μ(X)(!a.((?b.(0)) + (?c.(X)))),0)

Figure 8.3: Output of `ccs12.toDot()`.

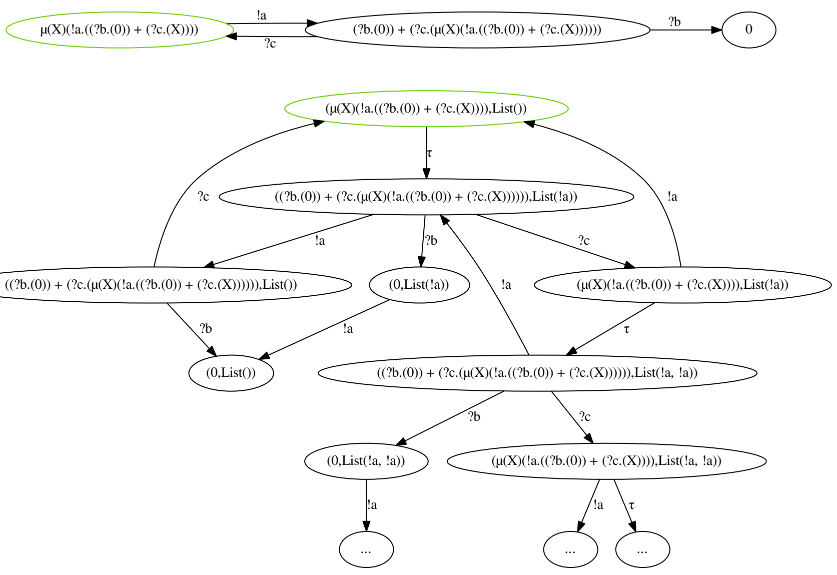

μ(X)(!a.((?b.(0)) + (?c.(X))))
!a
?c
(?b.(0)) + (?c.(μ(X)(!a.((?b.(0)) + (?c.(X))))))
?b
0
(μ(X)(!a.((?b.(0)) + (?c.(X)))),List())
τ
((?b.(0)) + (?c.(μ(X)(!a.((?b.(0)) + (?c.(X)))))),List(!a))
?c
!a
((?b.(0)) + (?c.(μ(X)(!a.((?b.(0)) + (?c.(X)))))),List())
(0,List(!a))
(μ(X)(!a.((?b.(0)) + (?c.(X)))),List(!a))
(0,List())
((?b.(0)) + (?c.(μ(X)(!a.((?b.(0)) + (?c.(X)))))),List(!a, !a))
(0,List(!a, !a))
(μ(X)(!a.((?b.(0)) + (?c.(X)))),List(!a, !a))
...
...
...

Figure 8.4: Output of `ccs1.toDot()` (top) and `ccs1a.toDot(maxDepth=Finite(4))` (bottom).

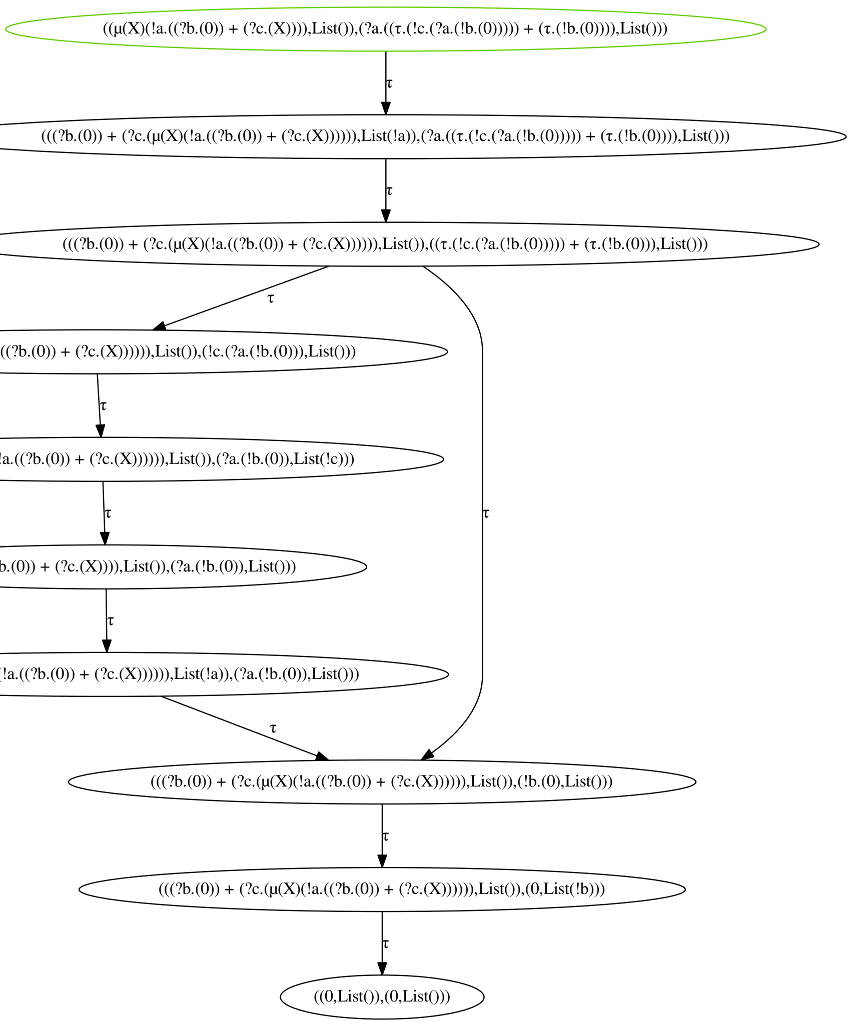

((μ(X)(!a.((?b.(0)) + (?c.(X)))),List()),(?a.((τ.(!c.(?a.(!b.(0))))) + (τ.(!b.(0))))),List()))
τ
(((?b.(0)) + (?c.(μ(X)(!a.((?b.(0)) + (?c.(X))))))),List(!a)),(?a.((τ.(!c.(?a.(!b.(0))))) + (τ.(!b.(0))))),List()))
τ
(((?b.(0)) + (?c.(μ(X)(!a.((?b.(0)) + (?c.(X))))))),List()),((τ.(!c.(?a.(!b.(0))))) + (τ.(!b.(0)))),List()))
τ
τ
(((?b.(0)) + (?c.(μ(X)(!a.((?b.(0)) + (?c.(X))))))),List()),(!c.(?a.(!b.(0)))),List()))
τ
(((?b.(0)) + (?c.(μ(X)(!a.((?b.(0)) + (?c.(X))))))),List()),(?a.(!b.(0))),List(!c)))
τ
((μ(X)(!a.((?b.(0)) + (?c.(X)))),List()),(?a.(!b.(0))),List()))
τ
(((?b.(0)) + (?c.(μ(X)(!a.((?b.(0)) + (?c.(X))))))),List(!a)),(?a.(!b.(0))),List()))
τ
(((?b.(0)) + (?c.(μ(X)(!a.((?b.(0)) + (?c.(X))))))),List()),(!b.(0)),List()))
τ
(((?b.(0)) + (?c.(μ(X)(!a.((?b.(0)) + (?c.(X))))))),List()),(0,List(!b)))
τ
((0,List()),(0,List()))

Figure 8.5: Output of `(ccs1a ||| ccs2a).toDot(filter={case (l,_) => l.isTau})`. Note that τ-transitions generated by synchronisations cause the reduction of buffers — i.e., the output at the head of a buffer is consumed by an input of the other process.

Chapter 9

Related work

This section discusses several related papers, grouping them by topic, and outlining some possible extensions to our theory.

9.1 Session types

Session types were introduced by Honda *et al.* in [Hon93; THK94; HVK98], as a type system for communication channels in a variant of the π-calculus. The resulting concept of *structured communication-based programming* has been the cornerstone on which a thriving research trend has been developed throughout the following decades.

In [MV11], session types are coupled with a "featherweight" Erlang-like language that, however, omits the problematic `receive`... `after`... construct discussed in Section 1.1.1. While adapting the type system of [MV11] to cope with such construct should be feasible, our approach allows the construction of the type system (in our case, the rules for $\ddot{\preccurlyeq}$) to be driven by an explicit underlying semantic notion (the I/O simulation).

9.2 Other "foundational" approaches to session types

A foundational approach to the theory of session types can be found in [Vas09], where session types and typing rules are gradually "reconstructed" in linear π-calculus. The similarity with the present work is mainly "moral", in the sense that both approaches start from a minimal setting, and then introduce and justify each element of the theory "as needed", instead postulating everything upfront. The main difference, though, is that [Vas09] is more syntax-oriented: for instance, it only provides (reduction) semantics for processes (and not types).

A different foundational approach is taken in [Cas+09], where types and processes have their specific language (equipped with higher-order LTS semantics) but most definitions are semantic-based, rather than syntax-oriented, to facilitate their adaptation and reuse in other calculi. It is possible to find several similarities between [Cas+09] and our work: the duality and subtyping/subsessioning relations introduced in the former (for which completeness and decidability results are also provided) are based on *"may output"* and *"must input"* relations, which embody input/output asymmetry — reminding the I/O asymmetry and the "persistent inputs" which are also found in our I/O compliance and I/O simulation; moreover, [Cas+09] implements a form of *"partial asynchrony"*, where output transitions are non-blocking and *"irrevocable"* (inspired by [CH98]). The main differences between our work and [Cas+09], instead, are that we forfeit all dependencies on any process/type language (by focusing on a first-order I/O LTS populated by *both* processes and types), and therefore our definitions must also take into account behaviours which are *not* constrained (in particular, in their τ-transitions and output irrevocability) by an underlying language with given internal/external choice operators. Moreover, within the same framework, we also embed behaviours with explicit FIFO buffers, thus obtaining a more extensive treatment of asynchrony. The similarities between the approaches, however, suggest that by embedding the types and processes of [Cas+09] in our I/O LTS, and focusing on them, we could be able to specialise (and simplify) several definitions and achieve similar results. This could allow to recover the completeness of $\ddot{\leqslant}$ w.r.t. $\sqsubseteq$ in further classes of behaviours populating $\mathbb{U}$, as discussed in Remark 6.17. In particular, the (semantic) notion of *"viable descriptor"* in [Cas+09] (Definition 2.9) and a semantic counterpart of the *"successor descriptor"* (Definition 2.4) could be the basis for a class of "viable behaviours" for which $\ddot{\leqslant}$ could coincide with $\sqsubseteq$.

9.3 Multiple participants and multiple sessions

Some recent works extend the session types discipline to the multiparty case, starting from [HYC08]. In this setting, the application designer specifies the overall communication behaviour of multiple participants through a *choreography*, which enjoys some correctness properties (e.g., safety and progress). The overall application is the result of the composition of a set of processes, which are distributed over the network and interact through a multiparty session. To ensure the correctness of this composition, the choreography is projected into a set of *local session types*, which abstract the end-point communication behaviour of processes: if each process is type-checked against its session type, the composition of services preserves the properties enjoyed by the choreography. The crucial technical difference w.r.t. dyadic session types is that, in *local* multiparty session types, input/output actions target the specific participant from/to which a message should be received/sent; correspondingly, a process typechecks only if its input/output actions target the correct participant(s).

Dyadic session types represent a particular case of multiparty session types with just two participants: in this case, a choreography is just a pair of types T, U such that $T \dashv\vdash U$ (i.e., $T \mathrel{\ddot{\bowtie}} U$, by Theorem 4.9), and the recipient/sender of each input/output action is always the other endpoint. We expect that our approach can be extended to this setting, too: some insights come from the streamlined approach of [CDP12], where the authors *"take a step back . . . defining global descriptions whose restrictions are semantically justified"*. The plan is to extend the $\ddot{\leqslant}$ relation to capture the *role* of each type/process, and then to produce the syntax-based typing rules via (partial) axiomatisation for a given calculus.

We also plan to address the orthogonal problem "correct" interactions in the presence of multiple *interleaved* sessions — that in the present work have not been addressed, as explained in Section 1.2. Two starting points are [Bar+13; PVV14], that introduce type systems for ensuring liveness in this setting.

9.4 Subtypes, subcontracts and sub-behaviours

[GH99; GH05] study subtyping for (dyadic) session types. These works focus on *communication channel* replaceability, and yield a relation which, following [PS96], is inverse w.r.t. $\ddot{\leqslant}$ in its handling of output/input covariance/contravariance. The intuition in such works is the following: consider a type U and its subtype T, and a program P interacting through a channel; then, a channel typed with T is "less demanding" for P w.r.t. a channel typed with U — i.e., if P correctly uses an U-typed channel, then it can also correctly use a T-typed channel. This holds because T allows P to choose among "more" possible outputs, and mandates "less" inputs to be enabled, w.r.t. U. From the other endpoint of a session, instead, the intuition is reversed: a subtyped channel allows a process P to have a "more demanding" behaviour — i.e., since T allows P to perform "more" outputs and allows for "less" inputs w.r.t. U, there are *fewer* processes which are I/O compliant with T w.r.t. U. In the present work, we look at behaviours as external observers, i.e. as if we were on the other endpoint of a channel; moreover, we want a subtype to be "less demanding" for the other endpoint, thus being I/O compliant with *more* behaviours. Therefore, our $\ddot{\leqslant}$-induced ordering intuitively works in the following way: we fix a channel type T, and given some program P which interacts according to T, we look for a "smaller" program P' which can replace P, without diverging from T. Then, we have that P' must choose among "less" possible outputs, and expose "more" inputs w.r.t. to P — i.e., the ordering is reversed.

[CGP09] introduces a subcontract relation whose direction is also opposite w.r.t. $\ddot{\leqslant}$, albeit for a different reason: given a contract with a set of compliant behaviours, a subcontract can correctly interact with a *subset* of such behaviours. The same principle governs the subtype/subsession relation in [Cas+09]. In these cases, a subcontract/subtype is "more demanding" than a supercontract/supertype. The intuition behind $\ddot{\leqslant}$, instead, is that we want a sub-behaviour to be "less demanding" w.r.t. a super-behaviour: i.e., if we fix a contract r and a super-behaviour p which is I/O compliant with it, then a sub-behaviour p' is I/O compliant with r, too (as per Definition 6.1).

A subtyping relation with the same ordering of $\ddot{\leqslant}$ is studied in [CHY07; DH11; CHY12]. There, a process P that uses a channel with subtype T can also use a channel with supertype U (i.e., the ordering of T and U is opposite w.r.t. [GH99; GH05]); this means that, if compared with U, the subtype T allows for "less" outputs and requires "more" inputs to P; this, in turn, means that from the other endpoint of a session, T is "less demanding" (i.e., is I/O compliant with more behaviours) than U. This matches the co/contravariance of outputs/inputs that $\ddot{\leqslant}$ induces on T and U.

The topics in [GH05] are reprised in [BL10; BL14; BL15], where session types are equipped with (first-order and higher-order) LTS semantics (by encoding them in a fragment of the contract language of [CGP09]), and studied under different notions of client-server compliance (e.g., allowing the client to terminate interaction or to skip messages). We took inspiration from these works, aiming at a framework general enough to faithfully replicate their notions and results. A similar line of research is also taken in [BH12; BH14], where one of the main results is the full abstraction of the encoding of types into contracts — which was missing in [BL10] (but later added in the long version [BL15]). This result is akin the (higher-order) combination of Lemma 6.6 and Theorem 6.16 (item (b)), that we proved independently. The main difference, however, is that we chose to focus on first-order behaviours in a completely language-independent treatment — whereas [BL10; BL14; BL15; BH12; BH14] focus on two specific languages (i.e., session types and a fragment of the contracts from [CGP09]), investigating the correspondence of syntactic and semantic definitions of subtyping in the two settings. The notion of "higher-order contract" adopted in these works, whose LTSs feature labels modelling input/output of contracts (which in turn have their own LTS semantics), could be a starting point for an higher-order extension of our I/O LTS.

9.5 Asynchrony and session types

Asynchronous dyadic session types have been addressed in [NT04], where type equivalence up-to buffering was defined over traces, and then approximated via syntax-based rules.

[MYH09; MY09; Mos09] study subtyping with "safe" partial commutativity allowed by asynchrony, in the setting of multiparty session types: these works also tackle possible optimisations allowed by buffering, similarly to our "asynchronous" Alice-Bartender interaction (Chapter 3).

These works have been recently reprised in [CDY14], where it is noted that the asynchronous subtyping from Mostrous *et al.* guarantees progress, but does not offer guarantees about orphan messages: in case of recursive types, a subtype may not expose an input which is infinitely often exposed in the supertype; thus, a program matching the subtype may "forget" to read some messages from its queues. We will further discuss this topic later, in Section 9.6.

In [KYH11] a bisimulation is defined to relate processes communicating via unbounded buffers.

9.6 Compliance and safety

A notion of compliance and refinement for services with asynchronous communication has been studied in [BZ08] (which extends [BZ07b]). There, service contracts are specified as processes in a (finite-state) variant of CCS with a special state indicating successful termination, and the semantics of (multi-party) systems of contracts is given by pairing such contracts with unbounded buffers. This approach is pretty similar to our pairing of session types and CCS terms with queues, as per Definitions 2.16 and 2.25 — and even closer to the "LTS-level asynchrony" we implemented in `LTSwb` (Section 8.1.3), where an asynchronous LTS is induced from a synchronous one. The main differences between the two works are that our I/O compliance and I/O simulation relations are coinductive (whereas the relations in [BZ08] are trace-based), we do not focus in terminating behaviours (i.e., we do not require a successful termination state to be always reachable), and that we embed the presence of buffers *within* the processes, thus making such a detail "invisible" for our behavioural relations (whereas [BZ08] adds explicit labels signalling enqueueing/dequeueing of messages, in the style of Communicating Finite State Machines).

Also [Pad10] addresses the problem of defining compliance between service contracts. In their *weak compliance* relation, finite-state orchestrators can resolve external choices or rearrange messages in order to guarantee progress. Weak compliance is unrelated to our I/O compliance: on the one hand, the latter cannot rearrange messages; on the other hand, I/O compliance has no fixed bound on the size of the buffers. For instance, let $!\mathsf{a}^m$ be a sequence of m $!\mathsf{a}$; the async behaviours $!\mathsf{a} \,.\, ?\mathsf{b} \,.\, !\mathsf{a}^2 \,.\, ?\mathsf{b}^2 \cdots !\mathsf{a}^n \,.\, ?\mathsf{b}^n \cdots$ and $!\mathsf{b} \,.\, ?\mathsf{a} \,.\, !\mathsf{b}^2 \,.\, ?\mathsf{a}^2 \cdots !\mathsf{b}^n \,.\, ?\mathsf{a}^n \cdots$ are I/O compliant, but they are not weakly compliant, as orchestrators must have a finite rank.

Compliance and safety are widely studied in the field of Communicating Finite State Machines (CFSMs) [BZ83]. In our setting, the pairing of a session type (or CCS process) with an output buffer is quite close to the pairing of a CFSM with its output buffer, albeit the resulting transition diagram is different: e.g., in our setting, an output is added to a buffer with a τ-transition, and consumed with a τ-synchronisation; in CFSMs, an output $!\mathsf{a}$ is buffered with a visible $!\mathsf{a}$-transition, and is consumed with a visible $?\mathsf{a}$-transition. Despite these differences, we adapted the usual notions of orphan message [LTY15; DY13] and unspecified reception [CF05; LTY15; DY13] in our setting. In the dyadic setting, our notion of I/O compliance is actually *stronger* than the notions of compliance introduced in [LTY15; DY13], essentially because, as discussed in Example 5.16, our notion of orphan message is stricter. For instance, consider:

$$T = \mathrm{rec}_X\, \mathsf{!a} \qquad U = \mathrm{rec}_Y\, \mathsf{!b}$$

The composition $T[\,] \parallel U[\,]$ is deadlock-free and does not reach unspecified reception configurations; by Definition 5.5, however, it is an orphan message configuration (because outputs are sent but never received), and therefore not safe. According to the definitions in [LTY15; DY13], instead, such a configuration is considered safe: in fact, a message is only considered "orphan" when the machines terminate with non-empty buffers.

A notion of "orphan message" closer to ours is studied in [CDY14]: there, a process "orphans" a buffered message when it has no possibility to read it in the future — even though the process itself is not terminated. A relevant difference w.r.t. our approach is that, in [CDY14], buffers are not introduced at the types level, but at the level of π-calculus processes — thus making the definition of error conditions more complex. Another difference is that [CDY14] defines an *asynchronous subtyping* which, roughly, allows to swap the order of input/output actions under certain conditions, whereas the basic duality between endpoint types is computed under the intuition of synchronous type semantics. In our approach, instead, subtyping (i.e., I/O simulation) is generally more restrictive in the reordering of input/output actions (even in presence of buffers), while I/O compliance is more flexible (see Examples 4.10 and 7.17). We conjecture that, by adapting the definitions of [CDY14] to our framework, it would be possible to find a strong connection between the combination of duality + asynchronous subtyping, and the combination of I/O compliance + I/O simulation: this investigation is left as future work.

The aim of Theorems 4.13 and 6.27 is to provide for a unifying approach to the issues discussed in Sections 9.5 and 9.6, by studying them in a remarkably simple and abstract setting, and tranferring properties from the synchronous to the asynchronous semantics.

9.7 Testing

Several works on contracts and session behaviours, e.g. [BZ07b; BZ08] denote the successful termination of a behaviour with a specific transition label (e.g. ✓) and/or a specific state (e.g. **1** or End). In this work, we consider two behaviours to be I/O compliant when they synchronise until the client (in the asymmetric case) simply stops interacting, i.e. reaches a state isomorphic to **0**. It is easy to extend our framework with a success label/state, thus allowing e.g. to study a testing theory [NH84]. For simplicity, we chose not to address such an extension in the present treatment, and leave it as future work. We conjecture that such an extension would allow to replicate the results of [BH13], which studies the difference between compliance preorders (where "compliance", in this case, is *progress*, as in Definition 4.1) and testing preorders, in several (synchronous and first-order) contract languages; a further natural development would be comparing such preorders with I/O compliance preorders, and further compare them with I/O simulation.

9.8 Abstracting richer calculi

As discussed in Section 1.2, this work approaches the session types theory through several abstractions, allowing us to focus on the fundamental behavioural theory.

Our approach shares some common ground with [CGP09; Car+06]: the inspiration to [DH87] for the (synchronous) session types semantics, the idea of representing processes and contracts/types in the same LTS, thus allowing for easy reasoning about their progress/compliance properties, and the will to overcome the rigid internal/external choices dichotomy required by session types, emerging both at type and process levels. In [CGP09], it is assumed that some type system can *abstract* processes P, Q (expressed in *any* calculus) into contracts. This type system must be *"consistent"* and *"informative"*, by preserving some essential properties like e.g. visible actions and internal non-determinism. A result in [CGP09] is that if the abstractions of P, Q are (strongly) compliant, then P, Q will be (strongly) compliant as well.

A similar abstraction is at the root of our approach. Our I/O LTS does not explicitly support *value passing* and *higher-order communication* (i.e., transmission of channels or processes), which are common features in most works on session types: this simplification, resulting in a remarkably streamlined setting, still allows us to study non-trivial synchronisation problems between (first-order) input/output capabilities, and to address asynchronous communication. We believe that enriching the labels of I/O LTSs to also cater for (higher-order) data transmission is feasible — but, according to our experience, we also believe that such a change would mostly address problems that are orthogonal w.r.t. the underlying behavioural theory. Instead, we believe that these issues can be better addressed by adapting to our framework the notion of consistent/informative abstractions of [CGP09]: it would allow, for instance, to abstract richer process calculi into an LTS populated with symbolic I/O sorts (like the one adopted in this work); the abstraction itself could be built upon the results presented e.g. in [MPS14], which studies the translation of higher-order LTSs into first-order ones.

Beyond these general ideas, the technical developments of this work are substantially different from [CGP09]: in the strong subcontract relation of [CGP09] there is no input/output distinction, and some desirable subtypings do not hold, e.g. $\mathsf{?a} \,\&\, \mathsf{?b} \not\sqsubseteq \mathsf{?a}$. These are restored through a "weak" subcontract relation, exploiting *filters* to suitably resolve external non-determinism. A challenging task would be that of using filters to enforce the I/O co/contra-variance typical of session types (and embodied in $\ddot{\bowtie}$ and $\ddot{\lessgtr}$), thus allowing to replicate our results in their framework. This appears technically complex, and is left as future work.

9.9 Timeouts and exceptions

Our running example in Chapter 3 shows a case in which a timeout event (abstracted as a τ-transition) interrupts an external choice. This can be seen as a particular instance of *exception handling* in session-based interactions, which has been addressed in [CGY10; Hu+13]. These works enrich the session types language with new constructs and messages, marking where an exceptional event might occur within a protocol, and how it can be handled at runtime; moreover, they introduce run-time monitoring to ensure proper coordination of the communicating processes. The present work does not aim at fully addressing these problems, and we only presented a case that can be addressed "locally" by a process, i.e. without requiring dedicated communications or runtime mechanisms. It is worth noticing, however, that our semantic framework allows to handle type languages which are more expressive than (binary) session types — and even the syntactic relation $\ddot{\preccurlyeq}_{\Gamma}$ (Definition 7.10) is defined on a fragment of CCS that, unlike session types, supports free τ-prefixes. We conjecture that, just as in our running example, several cases of "types with exceptions" can be modelled through choices among inputs/outputs with additional τ-branches representing "exceptional" events; therefore, they could be embedded and studied in our current framework without requiring significant extensions.

Chapter 10

Conclusions

We have revisited the theory of session types from a purely semantic perspective. We have defined the *I/O compliance* relation $\ddot{\bowtie}$, showing it to be sound and complete w.r.t. *safety* for asynchronous session types; we have defined an *I/O simulation* relation $\ddot{\leqslant}$ between generic behaviours, showing it to be a compliance-preserving preorder unifying the notions of typing and subtyping for session types, as well as their synchronous and asynchronous interpretations.

In this work we mostly focused on behaviours arising from (synchronous and asynchronous) session types and CCS; however, it seems that our framework can be easily exploited to analyse the properties of other behaviours populating $\mathbb{U}$ — e.g. the LTS semantics of other process calculi and programming languages.

10.1 Summary of the main results

We have shown that "client-biased" I/O compliance $\ddot{\lhd}$ implies "client-biased" progress $\dashv$ (and thus, for the symmetric versions, $\ddot{\bowtie}$ implies $\dashv\vdash$). We have also shown that, for synchronous session types ($\mathbb{U}_{\mathrm{ST}}$), the two relations coincide.

Theorem 4.9. *(a) If $p \mathrel{\ddot{\lhd}} q$, then $p \dashv q$; (b) if $p, q \in \mathbb{U}_{ST}$ and $p \dashv q$, then $p \mathrel{\ddot{\lhd}} q$.*

We have shown that, on general behaviours, I/O compliance implies safety, i.e. absence of deadlocks, orphan messages and unspecified reception configurations (as defined in Section 5.4). Moreover, the two relations coincide for asynchronous session types ($\mathbb{U}_{\mathrm{aST}}$). The following theorem summarises Theorems 5.22 and 5.24:

Theorem. *(a) If $p \mathrel{\ddot{\bowtie}} q$ then $p \parallel q$ is safe. (b) If $p, q \in \mathbb{U}_{aST}$ then $p \parallel q$ safe implies $p \mathrel{\ddot{\bowtie}} q$.*

We have shown that I/O simulation $\ddot{\leqslant}$ is an I/O compliance-preserving (and thus, safety-preserving) preorder for general behaviours. The following theorem summarises Theorems 6.12 and 6.13.

Theorem. *$\ddot{\leqslant}$ is a preorder, and $p \mathrel{\ddot{\leqslant}} q \circ r$ implies $p \circ r$, for $\circ \in \{\ddot{\rhd}, \ddot{\bowtie}\}$.*

We have shown that all the main relations presented in this work are preserved when passing from synchronous to asynchronous semantics of session types. The following theorem summarises Theorems 4.13 and 6.27:

Theorem. *For all session types T, U:*

- *If $T \circ U$, then $T[] \circ U[]$, for $\circ \in \{\vdash, \dashv\vdash, \dashv, \ddot{\rhd}, \ddot{\bowtie}, \ddot{\lhd}\}$.*
- *If $T \; (\ddot{\leqslant} \cap \preceq_{?}) \; U$, then $T[] \mathrel{\ddot{\leqslant}} U[]$.*

We have introduced a proof-of-concept typing system semantically grounded on $\ddot{\leqslant}$ and $\ddot{\bowtie}$, allowing to syntactically type our examples from Chapter 3 (which exploit asynchronous semantics and external choices mixed with τ-transitions).

Theorem 7.16 (Correctness)**.** *If $\vdash P : T$ and $\vdash Q : U$ with $T[] \mathrel{\ddot{\bowtie}} U[]$, then $P[] \mathrel{\ddot{\bowtie}} Q[]$.*

Finally, in Chapter 8, we have described `LTSwb`, an LTS manipulation library implementing part of the theory presented in this work, showing that it validates our main Example 7.18.

10.2 Future work

We now discuss some possible developments based on our semantic framework, thus integrating the future work already outlined in Chapter 9.

10.2.1 Some conjectures on $\ddot{\precsim}$

This section discusses some conjectured properties of $\ddot{\precsim}$. We start with Conjecture 10.1, which investigates the possibility of implementing our syntactic approximation of $\ddot{\leqslant}$.

Conjecture 10.1. *$\ddot{\precsim}$ is decidable.*

The main insight supporting Conjecture 10.1 is that, whenever $P \ddot{\precsim}_\Gamma Q$ with some premise $P' \ddot{\precsim}_\Gamma Q'$, the rules for $\ddot{\precsim}$ give the following relation:

$$P' \text{ is a proper subterm of } P \quad \text{or} \quad Q = \mu_X Q'' \text{ and } Q' = Q''[{}^{Q}/_{X}]$$

Such a relation seems to be a well-founded order. In fact, when applying the rules upwards, either P reduces in size, or Q is a recursive term whose top-level can only be unfolded for a *finite* number of times (since terms under recursion are sequential, recursion variables are guarded, and thus CCS$^-$ terms are contractive). Hence, after a finite number of steps,

- no rule can be applied, or
- P is reduced to a free variable or to $\mathbf{0}$, and is paired with a derivative of Q which cannot be unfolded further.

However, the missing piece for turning Conjecture 10.1 into a result is a careful usage of term substitutions via $\equiv$: in fact, if used freely, $\equiv$ can increase the size of terms (e.g., allowing to rewrite P into $P \mid \mathbf{0}$), thus breaking the ordering above.

Conjecture 10.2 below states that $(\mathbb{U}^{-}_{\mathrm{CCS}}, \ddot{\precsim})$ is a preorder that is preserved by all the operators of CCS$^-$, that is μ, $+$ and $\mid$ (whenever subterm substitutions result in valid CCS$^-$ processes).

Conjecture 10.2. *$\ddot{\precsim}$ is a precongruence for CCS$^-$.*

Proof. See page 156. The missing piece for this result is a proof for the conjecture: $\mu_X P \ddot{\precsim}_\Gamma Q$ implies $P[{}^{\mu_X P}/_{X}] \ddot{\precsim}_\Gamma Q$. □

A non-obvious aspect of Definition 7.10 and Conjecture 10.2 is that, by requiring guarded choices in CCS$^-$, $\ddot{\precsim}$ is preserved by $+$ (via rule (+Ctx)). This is *not* directly matched by a corresponding property for $\ddot{\lesssim}$ in full CCS without guarded choices: i.e. $P \ddot{\lesssim} Q \implies P+R \ddot{\lesssim} Q+R$. Indeed, the latter implication is false in general, because $\tau\,.\,?\mathsf{a}\,.\,P \ddot{\lesssim} ?\mathsf{a}\,.\,P$, but $?\mathsf{b} + \tau\,.\,?\mathsf{a}\,.\,P \not\ddot{\lesssim} ?\mathsf{b} + ?\mathsf{a}\,.\,P$. A similar argument holds for $\mid$, when arbitrary terms with overlapping inputs are put in parallel. This shows that $\ddot{\lesssim}$ is *not* a precongruence for CCS; by focusing on CCS$^-$ (where $\ddot{\preccurlyeq}$ is supposedly a precongruence), we can reason on whether two processes are $\ddot{\preccurlyeq}$-related by exploiting transitivity and substitution of $\ddot{\preccurlyeq}$-related sub-terms — without having to perform full derivations "from scratch".

10.2.2 Extending $\ddot{\preccurlyeq}$

We conclude this chapter with some comments about the possibility of extending $\ddot{\preccurlyeq}$, and thus the judgement $\vdash P : T$.

The addition of new rules is generally possible and pretty straightforward, under the approach we followed in Table 7.2: assuming some I/O simulations in the premises, and constructing a new one in the conclusions (with some additional care when the LHS "loses" branches w.r.t. the RHS, as in rule (+Int)). As an example, the "τ laws" from [Mil89] (§3.2) can be readily adopted. The main price for such an extension could be Conjecture 10.2: the "extended" $\ddot{\preccurlyeq}$ may not be a precongruence, and some adjustments may be needed if this (conjectured) result is deemed important. If it is *not* required, then it is possible, for instance, to drop condition *c.* in Definition 7.7, and replace rule (|LR) with the following, less restrictive one:

$$\frac{P \mathrel{\ddot{\leqslant}} Q \quad R \mathrel{\ddot{\leqslant}} S \quad \mathrm{ins}(P) \cap \mathrm{ins}(R) = \mathrm{ins}(Q) \cap \mathrm{ins}(S) = \emptyset}{P \mid R \mathrel{\ddot{\leqslant}} Q \mid S}\ (\mid\text{LR2})$$

i.e., two parallel branches are allowed to contain inputs, but the typing rule still ensures that they do not overlap.

Another possible extension to $\ddot{\preccurlyeq}$ is an additional (and more flexible) rule for "interruptible" external choices, such as:

$$\frac{\begin{array}{ll} Q \equiv \sum_{i\in I} ?\mathsf{a}_i \,.\, Q_i & K \neq \emptyset \\ \forall j \in J, i \in I \,.\, \mathsf{a}_j = \mathsf{a}_i \implies P_j \mathrel{\ddot{\leqslant}} Q_i & \forall k \in K \,.\, P_k \mathrel{\ddot{\leqslant}} Q \end{array}}{\sum_{j\in J} (?\mathsf{a}_j \,.\, P_j) + \sum_{k\in K} \tau \,.\, P_k \mathrel{\ddot{\leqslant}} Q}\ (+\text{Ext2})$$

The difference between (+Ext) and (+Ext2) is that the latter does *not* require the "immediate" input transitions of P to be a superset of those of Q — provided that the continuations of overlapping inputs are within the relation, and also the continuation after each τ-branch (of which at least one must exist) is within the relation. With such rule, the following bartender process has also type U_{B}:

$$\mu_Y\big(?\mathsf{aCoffee}\,.\,!\mathsf{coffee}\,.\,Y \;+\; \tau\,.\,\mu_Z(?\mathsf{aCoffee}\,.\,!\mathsf{coffee}\,.\,Z + ?\mathsf{aBeer}\,.\,!\mathsf{beer}\,.\,Z + ?\mathsf{pay})\big)$$

Here, the bartender does not want to sell beer before noon; however, he neither wants to upset the customers, which may go to another bar. Therefore, instead of accepting beer requests and answering $!\mathsf{no}$, he keeps serving coffee and postpone $?\mathsf{beer}$ and $?\mathsf{pay}$ requests — until he eventually decides (with a τ-transition) to also honour them.

Another topic worth discussing is condition *a.* of Definition 7.7, i.e. the restriction on inputs and branching under recursion in CCS$^-$. One of its main drawbacks is that some session types are *not* valid CCS$^-$ terms (see comment after Definition 7.13). As discussed on page 69, and earlier in Section 6.4, this restriction arises from the fact that I/O simulation preserves I/O compliance relations that hold in the *asynchronous* setting, but *not* in the synchronous one. Therefore, $\ddot{\precsim}$ and the judgement $\vdash P : T$ do *not* lead to safety results by assuming that T will be paired with some U (and some U-typed process) such that $T \mathrel{\ddot{\bowtie}} U$: such an assumption is typical in session types literature — but our *weaker* assumption in Theorem 7.16 is $T[] \mathrel{\ddot{\bowtie}} U[]$. Thus, since $\ddot{\leqslant}$ preserves the "strong" notion of orphan-message-freedom guaranteed by $\ddot{\bowtie}$ (see Section 9.6), we have to deal with the problematic cases described in Example 6.20.

However, a $\ddot{\leqslant}$-based type system can be developed in different ways, and condition *a.* of Definition 7.7 is not always required. We sketch two alternative approaches allowing to drop such a restriction.

Reducing asynchrony. We can restrict Theorem 7.16 to parallel compositions of processes whose types are *synchronously* I/O compliant. This would bring $\vdash P : T$ closer to the usual typing judgements from session types literature (except for the rules under the dashed line in Table 7.2, which are less common). However, this approach would preclude several results on our Alice/bartender use case — see discussion in Examples 7.17 and 7.18.

Axiomatising $(\ddot{\leqslant} \cap \preceq_{?})$. Instead of axiomatising $\ddot{\leqslant}$, we could study and axiomatise $(\ddot{\leqslant} \cap \preceq_{?})$ in CCS, aiming at a result similar to Theorem 6.27. Under this approach, it should be possible to develop a rule similar to (+Int), but with additional conditions on the output-guarded branches that the LHS can "lose" w.r.t. the RHS.

We expect that this would allow to obtain more general results, albeit at the price of an increased technical complexity.

Bibliography

[Bar+13] Massimo Bartoletti et al. “Honesty by Typing”. In: *FORTE*. 2013.

[BC08] Eduardo Bonelli and Adriana Compagnoni. “Multipoint Session Types for a Distributed Calculus”. In: *Trustworthy Global Computing*. Vol. 4912. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008. ISBN: 978-3-540-78662-7. DOI: 10.1007/978-3-540-78663-4_17.

[BCZ13] Massimo Bartoletti, Tiziana Cimoli and Roberto Zunino. “A theory of agreements and protection”. In: *POST*. 2013. DOI: 10.1007/978-3-642-36830-1_10.

[BDM12] Maria Grazia Buscemi, Rocco De Nicola and Hernán C Melgratti. “Contractual testing”. In: (2012). Unpublished.

[Bet+08] Lorenzo Bettini et al. “Global Progress in Dynamically Interleaved Multiparty Sessions”. In: *CONCUR*. 2008.

[BH12] Giovanni Bernardi and Matthew Hennessy. “Modelling Session Types Using Contracts”. In: *Proceedings of the 27th Annual ACM Symposium on Applied Computing*. SAC ’12. Trento, Italy: ACM, 2012, pp. 1941–1946. ISBN: 978-1-4503-0857-1. DOI: 10.1145/2245276.2232097.

[BH13] Giovanni Bernardi and Matthew Hennessy. “Compliance and Testing Preorders Differ”. In: *SEFM Workshops*. 2013.

[BH14] Giovanni Bernardi and Matthew Hennessy. “Using Higher-Order Contracts to Model Session Types (Extended Abstract)”. In: *CONCUR 2014 - Concurrency Theory - 25th International Conference, CONCUR 2014, Rome, Italy, September 2-5, 2014. Proceedings*. 2014, pp. 387–401. DOI: 10.1007/978-3-662-44584-6_27.

[BL10] Franco Barbanera and Ugo de'Liguoro. "Two Notions of Sub-behaviour for Session-based Client/Server Systems". In: *PPDP*. ACM SIGPLAN. ACM, 2010.

[BL14] Franco Barbanera and Ugo de'Liguoro. "Loosening the notions of compliance and sub-behaviour in client/server systems". In: *ICE*. 2014.

[BL15] Franco Barbanera and Ugo de'Liguoro. "Sub-behaviour relations for session-based client/server systems". In: *Mathematical Structures in Computer Science* FirstView (Feb. 2015), pp. 1–43. ISSN: 1469-8072. DOI: 10.1017/S096012951400005X.

[Boc+10] Laura Bocchi et al. "A theory of design-by-contract for distributed multiparty interactions". In: *CONCUR*. 2010.

[BSZ14] Massimo Bartoletti, Alceste Scalas and Roberto Zunino. "A Semantic Deconstruction of Session Types". In: *CONCUR 2014 - Concurrency Theory - 25th International Conference, CONCUR 2014, Rome, Italy, September 2-5, 2014. Proceedings*. 2014, pp. 402–418. DOI: 10.1007/978-3-662-44584-6_28.

[Bug+10] Michele Bugliesi et al. "Compliance Preorders for Web Services". In: *WS-FM*. 2010.

[BZ07a] Mario Bravetti and Gianluigi Zavattaro. "Contract Based Multi-party Service Composition". In: *International Symposium on Fundamentals of Software Engineering*. 2007. ISBN: 978-3-540-75697-2. DOI: 10.1007/978-3-540-75698-9_14.

[BZ07b] Mario Bravetti and Gianluigi Zavattaro. "Towards a Unifying Theory for Choreography Conformance and Contract Compliance". In: *Software Composition*. 2007.

[BZ08] Mario Bravetti and Gianluigi Zavattaro. "Contract Compliance and Choreography Conformance in the Presence of Message Queues". In: *WS-FM*. 2008.

[BZ83] Daniel Brand and Pitro Zafiropulo. "On Communicating Finite-State Machines". In: *J. ACM* 30.2 (Apr. 1983), pp. 323–342. ISSN: 0004-5411. DOI: 10.1145/322374.322380.

[Car+06] Samuele Carpineti et al. "A formal account of contracts for Web Services". In: *WS-FM*. 2006.

[Cas+09] Giuseppe Castagna et al. "Foundations of Session Types". In: *Proceedings of the 11th ACM SIGPLAN Conference on Principles and Practice of Declarative Programming*. PPDP '09. Coimbra, Portugal: ACM, 2009, pp. 219–230. ISBN: 978-1-60558-568-0. DOI: 10.1145/1599410.1599437.

[CDP12] Giuseppe Castagna, Mariangiola Dezani-Ciancaglini and Luca Padovani. "On Global Types and Multi-Party Session". In: *Logical Methods in Computer Science* 8.1 (2012).

[CDY07] Mario Coppo, Mariangiola Dezani-Ciancaglini and Nobuko Yoshida. "Asynchronous Session Types and Progress for Object Oriented Languages". In: *Formal Methods for Open Object-Based Distributed Systems (FMOODS)*. Vol. 4468. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2007. ISBN: 978-3-540-72919-8. DOI: 10.1007/978-3-540-72952-5_1.

[CDY14] Tzu-Chun Chen, Mariangiola Dezani-Ciancaglini and Nobuko Yoshida. "On the Preciseness of Subtyping in Session Types". In: *16th International Symposium on Principles and Practice of Declarative Programming*. ACM Press, 2014, pp. 146–135.

[CF05] Gérard Cécé and Alain Finkel. "Verification of programs with half-duplex communication". In: *Inf. Comput.* 202.2 (2005), pp. 166–190. DOI: 10.1016/j.ic.2005.05.006.

[CGP09] Giuseppe Castagna, Nils Gesbert and Luca Padovani. "A theory of contracts for Web services". In: *ACM TOPLAS* 31.5 (2009).

[CGY10] Sara Capecchi, Elena Giachino and Nobuko Yoshida. "Global Escape in Multiparty Sessions". In: *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010)*. Ed. by Kamal Lodaya and Meena Mahajan. Vol. 8. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2010, pp. 338–351. ISBN: 978-3-939897-23-1. DOI: http://dx.doi.org/10.4230/LIPIcs.FSTTCS.2010.338.

[CH98] Ilaria Castellani and Matthew Hennessy. "Testing Theories for Asynchronous Languages". English. In: *Foundations of Software Technology and Theoretical Computer Science*. Ed. by Vikraman Arvind and Sundar Ramanujam. Vol. 1530. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1998, pp. 90–101. ISBN: 978-3-540-65384-4. DOI: 10.1007/978-3-540-49382-2_9.

[CHY07] Marco Carbone, Kohei Honda and Nobuko Yoshida. “Structured Communication-Centred Programming for Web Services”. In: *Programming Languages and Systems, 16th European Symposium on Programming, ESOP 2007, Held as Part of the Joint European Conferences on Theory and Practics of Software, ETAPS 2007, Braga, Portugal, March 24 - April 1, 2007, Proceedings*. 2007, pp. 2–17. DOI: 10.1007/978-3-540-71316-6_2.

[CHY12] Marco Carbone, Kohei Honda and Nobuko Yoshida. “Structured Communication-Centered Programming for Web Services”. In: *ACM Trans. Program. Lang. Syst.* 34.2 (June 2012), 8:1–8:78. ISSN: 0164-0925. DOI: 10.1145/2220365.2220367.

[CP09a] Giuseppe Castagna and Luca Padovani. “A Preliminary Proposal of Decidable Testing Relations for Infinitary Asynchronous CCS”. In: (2009). Unpublished draft, presented at INFINITY’09.

[CP09b] Giuseppe Castagna and Luca Padovani. “Contracts for Mobile Processes”. In: *CONCUR*. 2009.

[CP10] Luís Caires and Frank Pfenning. “Session Types as Intuitionistic Linear Propositions”. In: *CONCUR*. 2010.

[CPS93] Rance Cleaveland, Joachim Parrow and Bernhard Steffen. “The Concurrency Workbench: A Semantics-based Tool for the Verification of Concurrent Systems”. In: *ACM Trans. Program. Lang. Syst.* 15.1 (Jan. 1993). ISSN: 0164-0925. DOI: 10.1145/151646.151648.

[CV10] Luís Caires and Hugo Torres Vieira. “Conversation types”. In: *Theor. Comput. Sci.* 411.51-52 (2010).

[DH11] Romain Demangeon and Kohei Honda. “Full Abstraction in a Subtyped pi-Calculus with Linear Types”. In: *CONCUR 2011 - Concurrency Theory - 22nd International Conference, CONCUR 2011, Aachen, Germany, September 6-9, 2011. Proceedings*. 2011, pp. 280–296. DOI: 10.1007/978-3-642-23217-6_19.

[DH87] Rocco De Nicola and Matthew Hennessy. “CCS without tau’s”. In: *TAPSOFT, Vol.1*. 1987.

[DY13] Pierre-Malo Deniélou and Nobuko Yoshida. "Multiparty Compatibility in Communicating Automata: Characterisation and Synthesis of Global Session Types". In: *ICALP*. 2013.

[Eri15] Ericsson Computer Science Laboratory. *The Erlang programming language*. http://erlang.org/. 2015.

[Gep15] Gephi developers community. *Gephi, the Open Graph Viz Platform*. 2015. URL: http://gephi.github.io/.

[GH05] Simon Gay and Malcolm Hole. "Subtyping for Session Types in the Pi Calculus". In: *Acta Inf.* 42.2 (2005). ISSN: 0001-5903. DOI: 10.1007/s00236-005-0177-z.

[GH99] Simon J. Gay and Malcolm Hole. "Types and Subtypes for Client-Server Interactions". In: *Programming Languages and Systems, 8th European Symposium on Programming, ESOP'99, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'99, Amsterdam, The Netherlands, 22-28 March, 1999, Proceedings*. 1999, pp. 74–90. DOI: 10.1007/3-540-49099-X_6.

[GV07] Simon Gay and Vasco T. Vasconcelos. *Asynchronous functional session types*. Tech. rep. 2007–251. University of Glasgow, May 2007.

[Hon93] Kohei Honda. "Types for Dyadic Interaction". In: *CONCUR*. 1993.

[Hu+13] Raymond Hu et al. "Practical Interruptible Conversations - Distributed Dynamic Verification with Session Types and Python". In: *Runtime Verification - 4th International Conference, RV 2013, Rennes, France, September 24-27, 2013. Proceedings*. 2013, pp. 130–148. DOI: 10.1007/978-3-642-40787-1_8. URL: http://dx.doi.org/10.1007/978-3-642-40787-1_8.

[HVK98] Kohei Honda, Vasco Thudichum Vasconcelos and Makoto Kubo. "Language Primitives and Type Discipline for Structured Communication-Based Programming". In: *ESOP*. 1998.

[HYC08] Kohei Honda, Nobuko Yoshida and Marco Carbone. "Multiparty asynchronous session types". In: *POPL*. 2008. DOI: 10.1145/1328438.1328472.

[KYH11] Dimitrios Kouzapas, Nobuko Yoshida and Kohei Honda. "On Asynchronous Session Semantics". In: *FMOODS/FORTE*. 2011.

[LTY15] Julien Lange, Emilio Tuosto and Nobuko Yoshida. "From communicating machines to graphical choreographies". In: *POPL 2015.* ACM, 2015, pp. 221–232.

[Mil89] Robin Milner. *Communication and concurrency.* Prentice-Hall, Inc., 1989.

[MK06] J Magee and J Kramer. *Concurrency - state models and Java programs (2. ed.).* LTS Analyser available at http://www.doc.ic.ac.uk/ltsa/. Wiley, 2006.

[Mos09] Dimitris Mostrous. "Session Types in Concurrent Calculi: Higher-Order Processes and Objects". PhD thesis. Imperial College London, Nov. 2009.

[MPS14] Jean-Marie Madiot, Damien Pous and Davide Sangiorgi. "Bisimulations Up-to: Beyond First-Order Transition Systems". English. In: *CONCUR 2014 – Concurrency Theory.* Ed. by Paolo Baldan and Daniele Gorla. Vol. 8704. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2014, pp. 93–108. ISBN: 978-3-662-44583-9. DOI: 10.1007/978-3-662-44584-6_8.

[MPW92] Robin Milner, Joachim Parrow and David Walker. "A Calculus of Mobile Processes, I and II". In: *Information and Computation* 100.1 (1992).

[MV11] Dimitris Mostrous and Vasco Thudichum Vasconcelos. "Session Typing for a Featherweight Erlang". In: *COORDINATION.* 2011.

[MY09] Dimitris Mostrous and Nobuko Yoshida. "Session-Based Communication Optimisation for Higher-Order Mobile Processes". In: *Proc. 9th International Conference on Typed Lambda Calculi and Applications (TLCA).* 2009, pp. 203–218. DOI: 10.1007/978-3-642-02273-9_16.

[MYH09] Dimitris Mostrous, Nobuko Yoshida and Kohei Honda. "Global Principal Typing in Partially Commutative Asynchronous Sessions". In: *Proc. 18th European Symposium on Programming (ESOP).* 2009, pp. 316–332. DOI: 10.1007/978-3-642-00590-9_23.

[NH84] Rocco De Nicola and Matthew Hennessy. "Testing equivalences for processes". In: *Theoretical Computer Science* 34.1–2 (1984), pp. 83–133. ISSN: 0304-3975. DOI: http://dx.doi.org/10.1016/0304-3975(84)90113-0.

[NT04] Matthias Neubauer and Peter Thiemann. "Session types for asynchronous communication". In: *Universität Freiburg* (2004).

[Oa04] Martin Odersky and al. *An Overview of the Scala Programming Language.* Tech. rep. IC/2004/64. Lausanne, Switzerland: EPFL, 2004. URL: http://scala-lang.org/.

[Pad10] Luca Padovani. "Contract-based discovery of Web services modulo simple orchestrators". In: *Theor. Comput. Sci.* 411.37 (2010).

[Pad12] Luca Padovani. "On Projecting Processes into Session Types". In: *Mathematical Structures in Computer Science* 22 (2 2012), pp. 237–289. ISSN: 0960-1295. DOI: 10.1017/S0960129511000405.

[Pad13] Luca Padovani. "Fair Subtyping for Open Session Types". English. In: *Automata, Languages, and Programming.* Ed. by FedorV. Fomin et al. Vol. 7966. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 373–384. ISBN: 978-3-642-39211-5. DOI: 10.1007/978-3-642-39212-2_34.

[Pie02] Benjamin Pierce. *Types and programming languages.* Cambridge, Mass: MIT Press, 2002. ISBN: 0262162091.

[PS96] Benjamin C. Pierce and Davide Sangiorgi. "Typing and Subtyping for Mobile Processes". In: *Mathematical Structures in Computer Science* 6.5 (1996), pp. 409–453.

[PVV14] Luca Padovani, Vasco Thudichum Vasconcelos and Hugo Torres Vieira. "Typing Liveness in Multiparty Communicating Systems". In: *COORDINATION.* 2014.

[San12] Davide Sangiorgi. *An introduction to bisimulation and coinduction.* Cambridge, UK New York: Cambridge University Press, 2012. ISBN: 1107003636.

[SB15] Alceste Scalas and Massimo Bartoletti. "The LTS WorkBench". In: *Proc. ICE.* To appear. 2015.

[Sca15] Alceste Scalas. *The LTS WorkBench (download page).* 2015. URL: http://tcs.unica.it/software/ltswb.

[SW01] Davide Sangiorgi and David Walker. *The π-calculus: A Theory of Mobile Processes.* Cambridge University Press, 2001.

[THK94] Kaku Takeuchi, Kohei Honda and Makoto Kubo. "An Interaction-based Language and its Typing System". In: *PARLE.* 1994.

[Vas09] VascoT. Vasconcelos. "Fundamentals of Session Types". English. In: *Formal Methods for Web Services*. Ed. by Marco Bernardo, Luca Padovani and Gianluigi Zavattaro. Vol. 5569. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, pp. 158–186. ISBN: 978-3-642-01917-3. DOI: 10.1007/978-3-642-01918-0_4.

[VVR06] Antonio Vallecillo, Vasco T. Vasconcelos and António Ravara. "Typing the Behavior of Software Components Using Session Types". In: *Fundam. Inf.* 73.4 (Sept. 2006). ISSN: 0169-2968.

[Wad12] Philip Wadler. "Propositions as sessions". In: *ICFP*. 2012.

Appendix A

Behaviours

A.1 Session types

Definition A.1 (Session types equivalence). *$\equiv$ is the relation between session types inductively defined by the rules:*

$$\mathbf{0} \equiv \mathbf{0} \;(\textsc{Eq-}\mathbf{0}) \qquad X \equiv X \;(\textsc{Eq-Var})$$

$$\mathrm{rec}_X\, T \equiv T\big[{}^{\mathrm{rec}_X\, T}/_X\big] \;(\textsc{Eq-UnfoldR}) \qquad T\big[{}^{\mathrm{rec}_X\, T}/_X\big] \equiv \mathrm{rec}_X\, T \;(\textsc{Eq-UnfoldL})$$

$$\frac{\forall i \in I \,.\, T_i \equiv T'_i}{\&_{i\in I} ?\mathsf{a}_i.T_i \equiv \&_{i\in I} ?\mathsf{a}_i.T'_i}\;(\textsc{Eq-Ctx}\&) \qquad \frac{\forall i \in I \,.\, T_i \equiv T'_i}{\bigoplus_{i\in I} !\mathsf{a}_i.T_i \equiv \bigoplus_{i\in I} !\mathsf{a}_i.T'_i}\;(\textsc{Eq-Ctx}\oplus)$$

$$\frac{T \equiv T'}{\mathrm{rec}_X\, T \equiv \mathrm{rec}_X\, T'}\;(\textsc{Eq-CtxRec}) \qquad \frac{T \equiv U \quad U \equiv V}{T \equiv V}\;(\textsc{Eq-Trans})$$

Proposition A.2. *$\equiv$ is a congruence relation for session types.*

Proof. Symmetry of $\equiv$ follows from the symmetry of the rules in Definition A.1. Transitivity holds by (Eq-Trans). Reflexivity is proved by structural induction on T, showing that a derivation for $T \equiv T$ can be obtained via rules (Eq-$\mathbf{0}$), (Eq-Var), (Eq-Ctx$\&$), (Eq-Ctx$\oplus$) or (Eq-CtxRec). Congruence for $\&$, $\oplus$ and rec follows respectively from rules (Eq-Ctx$\&$), (Eq-Ctx$\oplus$) and (Eq-CtxRec). □

Proposition 2.18. *For all T:*

(i) $T \xrightarrow{?\mathsf{a}}$ *iff* $T \equiv ?\mathsf{a}.T' \;\&\; \&_{i\in I} ?\mathsf{b}_i.T_i$;

(ii) $T \xrightarrow{\tau}$ *iff* $T \equiv \bigoplus_{i\in I} !\mathsf{b}_i.T_i$, *with* $|I| > 1$;

(iii) $T \xrightarrow{!\mathsf{a}}$ *iff* $T \equiv !\mathsf{a}.T'$.

Proof. For all items *(i)*–*(iii)*, the $\Longleftarrow$ direction follows from Definition 2.15.

For the $\Longrightarrow$ direction, we proceed by induction on the rules in Definition 2.15 generating the transitions:

- item *(i)*. The $?\mathsf{a}$-transition of T can be generated in two ways:

  a. by rule (TExt) (base case). Then, $T = ?\mathsf{a}.T' \mathbin{\&} \&_{i\in I} ?\mathsf{b}_i.T_i$. We conclude by reflexivity of $\equiv$ (Proposition A.2);

  b. by rule (TRec) (inductive case). Then, $T = \mathrm{rec}_X\, T''$, with premise $T''[^{\mathrm{rec}_X\, T''}/_X] \xrightarrow{?\mathsf{a}}$. By the induction hypothesis, we have $T''[^{\mathrm{rec}_X\, T''}/_X] \equiv ?\mathsf{a}.T' \mathbin{\&} \&_{i\in I} ?\mathsf{b}_i.T_i$; by (Eq-UnfoldR), we have $T \equiv T''[^{\mathrm{rec}_X\, T''}/_X]$ — and we conclude by transitivity of $\equiv$ (Proposition A.2);

- item *(ii)*. The τ-transition of T can be generated in two ways:

  a. by rule (TInt) (base case). Then, $T = \bigoplus_{i\in I} !\mathsf{b}_i.T_i$, with $|I| > 1$. We conclude by reflexivity of $\equiv$ (Proposition A.2);

  b. by rule (TRec) (inductive case). Then, $T = \mathrm{rec}_X\, T''$, with premise $T''[^{\mathrm{rec}_X\, T''}/_X] \xrightarrow{\tau}$. By the induction hypothesis, we have $T''[^{\mathrm{rec}_X\, T''}/_X] \equiv \bigoplus_{i\in I} !\mathsf{b}_i.T_i$, with $|I| > 1$; by (Eq-UnfoldR), we have $T \equiv T''[^{\mathrm{rec}_X\, T''}/_X]$ — and we conclude by transitivity of $\equiv$ (Proposition A.2);

- item *(iii)*. The $!\mathsf{a}$-transition of T can be generated in two ways:

  a. by rule (TOut) (base case). Then, $T = !\mathsf{a}.T'$. We conclude by reflexivity of $\equiv$ (Proposition A.2);

  b. by rule (TRec) (inductive case). Then, $T = \mathrm{rec}_X\, T''$, with premise $T''[^{\mathrm{rec}_X\, T''}/_X] \xrightarrow{!\mathsf{a}}$. By the induction hypothesis, we have $T''[^{\mathrm{rec}_X\, T''}/_X] \equiv !\mathsf{a}.T'$; by (Eq-UnfoldR), we have $T \equiv T''[^{\mathrm{rec}_X\, T''}/_X]$ — and we conclude by transitivity of $\equiv$ (Proposition A.2).

□

Proposition 2.19. *For all* T, T' *and* σ,

(i) $T \xrightarrow{?\mathsf{a}} T'$ *iff* $T[\sigma] \xrightarrow{?\mathsf{a}} T'[\sigma]$;

(ii) *for all* $\mathsf{a} \neq \mathsf{b}$, $T \xrightarrow{\tau} !\mathsf{a}.T'$ *and* $T \xrightarrow{\tau} !\mathsf{b}.T''$ *iff* $T[\sigma] \xrightarrow{\tau} T'[\sigma\,.!\mathsf{a}] \wedge T[\sigma] \xrightarrow{\tau} T''[\sigma\,.!\mathsf{b}]$;

(iii) $T \xrightarrow{!\mathsf{a}} T'$ *iff* $T[\sigma] \xrightarrow{\tau} T'[\sigma\,.!\mathsf{a}] \wedge \nexists \mathsf{b} \neq \mathsf{a}\,.\, T[\sigma] \xrightarrow{\tau} T'[\sigma\,.!\mathsf{b}]$.

Proof. $\Longrightarrow$ direction:

- item *(i)*: by Proposition 2.18 (item *(i)*), T is equivalent to a (possibly recursive) external choice with a $?\mathsf{a}$-branch. From such T, by rules (ruleCExtA) and (TRecA), we obtain the thesis;
- item *(ii)*: by Proposition 2.18 (item *(ii)*), T is equivalent to a (possibly recursive) internal choice with 2 or more branches. From such T, by rules (ruleCIntA) and (TRecA), we obtain the thesis, with $!\mathsf{a}, !\mathsf{b}$ being the distinct guards of two of the branches of T;
- item *(iii)*: by Proposition 2.18 (item *(iii)*), T is equivalent to a (possibly recursive) single-branch internal choice $!\mathsf{a}.T''$. From such T, by rules (ruleCIntA) and (TRecA), we obtain the thesis.

For the $\Longleftarrow$ direction, we proceed by induction on the rules in Definition 2.16 generating the transitions. The development is similar to the $\Longrightarrow$ direction in the proof of Proposition 2.18: we determine the possible syntactic form of T, which may be (up-to recursion) an external choice for item *(i)*, or an internal choice for items *(ii)*–*(iii)* (resp. with multiple branches including $!\mathsf{a}$ and $!\mathsf{b}$, or with just one $!\mathsf{a}$-branch). Then, we conclude by the rules in Definition 2.15 (i.e., (TExt) for item *(i)*, (TInt) for item *(ii)*, and (TOut) for item *(iii)*). □

Proposition 2.21. *$T[\sigma]\xrightarrow{!\mathsf{a}}$ iff $\exists \sigma' \,.\, \sigma = !\mathsf{a}.\sigma'$. If $T \overset{!\mathsf{b}}{\Rightarrow} T'$, then $\exists \mathsf{a}\,.\, T[\sigma] \xrightarrow{\tau} T'[\sigma\,.!\mathsf{b}]\xrightarrow{!\mathsf{a}}$.*

Proof. The first part of the statement follows by Definition 2.16. For the second part, since $T \overset{!\mathsf{b}}{\Rightarrow} T'$, then by Definition 2.15 it must be $T \equiv !\mathsf{b}.T' \oplus \bigoplus_{i\in I} !\mathsf{c}_i.T_i$ (and in particular, $I = \emptyset$ iff $T \xrightarrow{!\mathsf{b}} T'$). We then conclude by Definition 2.16. □

Proposition 2.22. *$T[\sigma]\overset{!!\mathsf{a}}{\Longrightarrow}$ iff $\exists \sigma' \,.\, \sigma = !\mathsf{a}.\sigma'$, or $\sigma = \epsilon$ and $\exists T' \,.\, T \equiv !\mathsf{a}.T'$*

Proof. For the $\Longleftarrow$ direction, there are the following two cases:

- $\sigma = \mathsf{!a}.\sigma'$. By Proposition 2.21, whenever $T[\mathsf{!a}.\sigma'] \Rightarrow T'[\sigma'']$, each τ-transition corresponds to an output of T being enqueued to the buffer. Therefore, $\sigma'' = \mathsf{!a}.\sigma'.\sigma'''$ (for some σ'''), and we have $T'[\sigma'']\xrightarrow{\mathsf{!a}}$, from which the thesis follows.

- $\sigma = \epsilon$ and $T \equiv \mathsf{!a}.T'$. The thesis follows from Definition 2.16.

For the $\Longrightarrow$ direction, we proceed by cases on the structure of σ:

- $\sigma = \mathsf{!a}.\sigma'$. The thesis follows trivially;

- $\sigma = \mathsf{!b}.\sigma'$ (with $\mathsf{b} \neq \mathsf{a}$). In this case, by Definition 2.16 we have $T[\sigma]\overset{\mathsf{!a}}{\not\Rightarrow}$ — and therefore, $T[\sigma]\overset{\mathsf{!!a}}{\not\Rightarrow}$ (contradiction);

- $\sigma = \epsilon$. Then either:

  - $T \equiv \&_{i\in I}\mathsf{?b}_i.T_i$. Then, by Definition 2.16, $\forall \mathsf{c} \,.\, T[\sigma]\overset{\mathsf{!c}}{\not\Rightarrow}$ — and therefore, $T[\sigma]\overset{\mathsf{!!a}}{\not\Rightarrow}$ (contradiction);

  - $T \equiv \bigoplus_{i\in I}\mathsf{!b}_i.T_i$. If $|I| \neq 1$, we have either:

    * $|I| = 0$. Then, $T = \mathbf{0}$ (since $I = \emptyset$), and therefore $T[\epsilon]\overset{\mathsf{!!a}}{\not\Rightarrow}$ (contradiction);

    * $|I| > 1$. Then, since the guards of $\oplus$ are pairwise distinct, $\exists i \in I \,.\, \mathsf{b}_i \neq \mathsf{a}$. Hence, $T[\epsilon] \xrightarrow{\tau} T_i[\mathsf{!b}_i]\overset{\mathsf{!a}}{\not\Rightarrow}$ (by Definition 2.16), and therefore $T[\sigma]\overset{\mathsf{!!a}}{\not\Rightarrow}$ (contradiction).

  We are left to examine the case $|I| = 1$: we have $T[\epsilon] \equiv \mathsf{!b}.T'[\epsilon] \xrightarrow{\tau} T'[\mathsf{!b}]$ for some $\mathsf{b} \in I = \{\mathsf{b}\}$ and T'. If $\mathsf{b} \neq \mathsf{a}$, by Definition 2.16 we have $T'[\mathsf{!b}]\overset{\mathsf{!a}}{\not\Rightarrow}$, and thus $T[\sigma]\overset{\mathsf{!!a}}{\not\Rightarrow}$ (contradiction). Therefore, we conclude that $T \equiv \mathsf{!a}.T'$ (for some T'). □

Appendix B

Compliance

Proposition B.1. *Let $p \circ q$, with $\circ \in \{\ddot{\lhd}, \ddot{\rhd}, \ddot{\bowtie}\}$. Then, $p \Rightarrow p'$ implies $p' \circ q$.*

Proof. We first prove the statement for $\circ = \ddot{\lhd}$. We proceed by induction on the length of the sequence of τ-transitions in $p \Rightarrow p'$. The base case ($n = 0$) follows from the hypothesis. For the inductive case, let $p \Rightarrow p^* \xrightarrow{\tau} p'$: by the induction hypothesis, we have $p^* \ddot{\lhd} q$ — and we conclude by item *c.* of Definition 4.4.

The proof for $\circ = \ddot{\rhd}$ is similar, except that we conclude by item *d.* of Definition 4.4.

Finally, the thesis for $\circ = \ddot{\bowtie}$ follows from Lemma 4.5. □

Proposition B.2. *Let $p \circ q$, with $\circ \in \{\ddot{\lhd}, \ddot{\rhd}, \ddot{\bowtie}\}$. Then, $p \overset{\mathsf{!a}}{\Longrightarrow} p' \wedge q \overset{\mathsf{?a}}{\Longrightarrow} q'$ implies $p' \circ q'$.*

Proof. Let:

$$\begin{array}{ccccccc} p & \Rightarrow & p_0 & \xrightarrow{\mathsf{!a}} & p_0' & \Rightarrow & p' \\ q & \Rightarrow & q_0 & \xrightarrow{\mathsf{?a}} & q_0' & \Rightarrow & q' \end{array}$$

We first prove the statement for $\circ = \ddot{\lhd}$. By applying Proposition B.1 on $p \Rightarrow p_0$, we have $p_0 \ddot{\lhd} q$; then, by applying Proposition B.1 on $q \Rightarrow q_0$, we have $p_0 \ddot{\lhd} q_0$. Now, by item *b.* of Definition 4.4), we obtain $p_0' \ddot{\lhd} q_0'$; finally, again by applying Proposition B.1 on $p_0' \Rightarrow p'$ and then on $q_0' \Rightarrow q'$, we conclude $p' \ddot{\lhd} q'$.

The proof for $\circ = \ddot{\rhd}$ is similar.

Finally, the thesis for $\circ = \ddot{\bowtie}$ follows from Lemma 4.5. □

Proposition B.3. *Let $p \ddot{\lhd} q$ and $p\overset{\mathsf{!a}}{\Longrightarrow}$. Then, $q\overset{\mathsf{??a}}{\Longrightarrow}$.*

Proof. Let q' be such that $q \Rightarrow q'$. By Proposition B.1 we have that $p \mathrel{\ddot{\lhd}} q'$; thus, by item *a.* of Definition 4.4, it follows that $p{\Downarrow}^{!} \subseteq \mathrm{co}(q'{\Downarrow}^{?})$. Therefore, $q' \overset{?\mathsf{a}}{\Rightarrow}$ — from which we conclude that $q \xRightarrow{??\mathsf{a}}$. □

Corollary B.4. *Let $w \in (\mathsf{A}^{!})^{*}$, and let $p \overset{w}{\Rightarrow}$. Then, $p \mathrel{\ddot{\lhd}} q$ implies $q \xRightarrow{\mathrm{co}(w)}$. Moreover, $\forall p', q' \,.\, p \overset{w}{\Rightarrow} p' \,\wedge\, q \xRightarrow{\mathrm{co}(w)} q'$ implies $p' \mathrel{\ddot{\lhd}} q'$.*

Proof. The first part of the statemen is proved by induction on w, and follows from Proposition B.1, Proposition B.3 and Proposition B.2.

The *"moreover..."* part, again by induction on w, follows from Proposition B.3. □

Proposition B.5. *Let $p \mathrel{\ddot{\lhd}} q$, with $p{\Downarrow}^{!} = \emptyset$, $p{\Downarrow}^{?} \neq \emptyset$ and $q \overset{!\mathsf{a}}{\Rightarrow}$: then, $p \xRightarrow{??\mathsf{a}}$.*

Proof. $\forall p' \,.\, p \Rightarrow p'$, by Proposition B.1 we have $p{\Downarrow}^{!} = \emptyset$ and $p' \mathrel{\ddot{\lhd}} q$; hence, by item *a.* of Definition 4.4), we have $\forall p' \,.\, p \Rightarrow p'$ implies $q{\Downarrow}^{!} \subseteq \mathrm{co}(p'{\Downarrow}^{?})$, i.e. $p' \overset{?\mathsf{a}}{\Rightarrow}$. We conclude that $p \xRightarrow{??\mathsf{a}}$. □

Proof of Lemma 4.5 on page 26

Proof. By Definition 4.4, we have $\ddot{\bowtie} \subseteq (\ddot{\rhd} \cap \ddot{\lhd})$. To prove the inverse inclusion, let $\mathcal{R} = (\ddot{\rhd} \cap \ddot{\lhd})$. Whenever $(p, q) \in \mathcal{R}$, we have $p \mathrel{\ddot{\rhd}} q$ and $q \mathrel{\ddot{\lhd}} p$, and:

a. $\big(p{\Downarrow}^{!} \subseteq \mathrm{co}(q{\Downarrow}^{?}) \;\wedge\; (p{\Downarrow}^{!} = \emptyset \wedge p{\Downarrow}^{?} \neq \emptyset \implies \emptyset \neq q{\Downarrow}^{!} \subseteq \mathrm{co}(p{\Downarrow}^{?}))\big)$ and
$\big(q{\Downarrow}^{!} \subseteq \mathrm{co}(p{\Downarrow}^{?}) \;\wedge\; (q{\Downarrow}^{!} = \emptyset \wedge q{\Downarrow}^{?} \neq \emptyset \implies \emptyset \neq p{\Downarrow}^{!} \subseteq \mathrm{co}(q{\Downarrow}^{?}))\big)$;

b. $p \xrightarrow{\ell} p' \wedge q \xrightarrow{\mathrm{co}(\ell)} q' \implies p' \mathrel{\ddot{\rhd}} q'$ and $q \xrightarrow{\ell'} q' \wedge p \xrightarrow{\mathrm{co}(\ell')} p' \implies q' \mathrel{\ddot{\lhd}} p'$;

c. $p \xrightarrow{\tau} p' \implies p' \mathrel{\ddot{\rhd}} q$ and $q \xrightarrow{\tau} q' \implies q' \mathrel{\ddot{\lhd}} p$;

d. $q \xrightarrow{\tau} q' \implies p \mathrel{\ddot{\rhd}} q'$ and $p \xrightarrow{\tau} p' \implies q \mathrel{\ddot{\lhd}} p'$.

With some simplifications and rearrangements, for all $(p, q) \in \mathcal{R}$, we have:

a. $p{\Downarrow}^{!} \subseteq \mathrm{co}(q{\Downarrow}^{?})$ and $q{\Downarrow}^{!} \subseteq \mathrm{co}(p{\Downarrow}^{?})$
and $(p{\Downarrow}^{!} = \emptyset \wedge p{\Downarrow}^{?} \neq \emptyset \implies \emptyset \neq q{\Downarrow}^{!} \subseteq \mathrm{co}(p{\Downarrow}^{?}))$
and $(q{\Downarrow}^{!} = \emptyset \wedge p{\Downarrow}^{?} \neq \emptyset \implies \emptyset \neq p{\Downarrow}^{!} \subseteq \mathrm{co}(q{\Downarrow}^{?}))$;

b. $p \xrightarrow{\ell} p' \wedge q \xrightarrow{\mathrm{co}(\ell)} q' \implies p' \mathrel{\ddot{\rhd}} q' \wedge q' \mathrel{\ddot{\lhd}} p'$, i.e. $p' \,\mathcal{R}\, q'$;

c. $p \xrightarrow{\tau} p' \implies p' \ddot{\rhd}\, q \wedge q\, \ddot{\lhd}\, p'$, i.e. $p'\,\mathcal{R}\,q$;

d. $q \xrightarrow{\tau} q' \implies p\, \ddot{\rhd}\, q' \wedge q'\, \ddot{\lhd}\, p$, i.e. $p\,\mathcal{R}\,q'$.

Therefore, each $(p,q) \in \mathcal{R}$ satisfies items *a.–d.* of Definition 4.4 — hence, $\mathcal{R}$ is an I/O compliance relation. Furthermore, $\mathcal{R}$ is symmetric. Thus, $\mathcal{R} = (\ddot{\rhd} \cap \ddot{\lhd}) \subseteq \ddot{\bowtie}$. □

Notation B.6. *We write $\sigma\,.T'$ for the session type obtained by prefixing T' with the sequence of outputs σ.*

Lemma B.7 (Sync session behaviours and async I/O compliance (I))**.** *Let $T \circ U$, for some $\circ \in \{\vdash, \dashv, \dashv\vdash, \ddot{\rhd}, \ddot{\lhd}, \ddot{\bowtie}\}$. Then, $T[] \Rightarrow T'[\sigma]$ implies $\sigma\,.T' \circ U$.*

Proof. We first prove the statement for $\circ = \dashv$, and therefore we show that:

$$T \dashv U \ \wedge\ T[] \Rightarrow T'[\sigma] \quad \text{implies} \quad \sigma\,.T' \dashv U \tag{B.1}$$

If T is equivalent to a (possibly empty) *external* choice, we have $T[] \overset{\tau}{\not\to}$, and hence $T' = T$ and $\sigma = \epsilon$: therefore, the thesis coincides with the hypothesis.

Otherwise, if T is a non-empty *internal* choice, we proceed by induction on the length of σ, which (by the semantics in Definition 2.16) corresponds to the number of τ-transitions along $T[] \Rightarrow T'[\sigma]$:

- base case: $\sigma = \epsilon$. Then, $T' = T$ and $\sigma = \epsilon$, and therefore the thesis coincides with the hypothesis;
- inductive case: $\sigma = \sigma'.!\mathsf{a}$, with $\sigma' = !\mathsf{b}_1.\ldots.!\mathsf{b}_\mathsf{n}$. In this case, we have $T[] \Rightarrow T''[\sigma'] \xrightarrow{\tau} T'[\sigma'.!\mathsf{a}]$, and by applying the induction hypothesis, $\sigma' T'' \dashv U$. Therefore, by applying Proposition 4.3 for n times along the synchronisations of $\sigma'.T''$ and U, we have:

$$\begin{array}{rcccl}
!\mathsf{b}_1.!\mathsf{b}_2.\ldots.!\mathsf{b}_\mathsf{n}.T'' & \dashv & U = U_1 & = & ?\mathsf{b}_1.U_2 \,\&\, \&_{i\in I_1} ?\mathsf{b}_\mathsf{i}.U_i \\
!\mathsf{b}_2.\ldots.!\mathsf{b}_\mathsf{n}.T'' & \dashv & U_2 & = & ?\mathsf{b}_2.U_3 \,\&\, \&_{i\in I_2} ?\mathsf{b}_\mathsf{i}.U_i \\
 & & \ldots & & \\
!\mathsf{b}_\mathsf{n}.T'' & \dashv & U_n & = & ?\mathsf{b}_\mathsf{n}.U_{n+1} \,\&\, \&_{i\in I_n} ?\mathsf{b}_\mathsf{i}.U_i \\
T'' & \dashv & U_{n+1} & &
\end{array}$$

At this point, since $T''[\sigma'] \xrightarrow{\tau} T'[\sigma'.!\mathsf{a}]$ (i.e., T'' enqueues an output in its buffer), by the semantics in Definition 2.16 we have:

$$T''[\sigma'] = \left(!\mathsf{a}.T' \oplus \bigoplus_{i \in I} !\mathsf{a}_i.T_i \right) [\sigma'] \xrightarrow{\tau} T'[\sigma'.!\mathsf{a}]$$

i.e., T'' must be a non-empty internal choice with a $!\mathsf{a}$-guarded branch; correspondingly, from $T'' \dashv U_{n+1}$ and Proposition 4.3, we have:

$$U_{n+1} = ?\mathsf{a}.U' \mathbin{\&} \mathop{\Large\&}_{k \in K} ?\mathsf{a}_k.U'_k \quad \text{with } I \subseteq K \text{ and } T' \dashv U'$$

But then, $\sigma'.!\mathsf{a}.T' = \sigma.T' \dashv U$. This concludes the proof of Equation (B.1).

With the proof above, the other possible values of $\circ$ can be verified in a straightforward way:

- case $\circ = \vdash$ can be proved in a similar way, noticing that in the inductive case, U may also become $\mathbf{0}$, i.e. it may terminate at any point without consuming T's output with an external choice;
- case $\circ = \dashv\vdash$ holds because (by Definition 4.1) $\dashv\vdash = \vdash \cap \dashv$;
- case $\circ = \ddot{\lhd}$ follows by Equation (B.1) and Theorem 4.9; case $\circ = \ddot{\rhd}$ is similar;
- finally, case $\circ = \ddot{\bowtie}$ follows by the previous item, because (by Lemma 4.5) $\ddot{\bowtie} = \ddot{\rhd} \cap \ddot{\lhd}$.

$\square$

Notation B.8. *Given a sequence of outputs $\sigma = !\mathsf{a}_1.\ldots.!\mathsf{a}_n$, we write $\xRightarrow{\sigma}$ for $\xRightarrow{!\mathsf{a}_1} \cdots \xRightarrow{!\mathsf{a}_n}$.*

Lemma B.9 (Diamond lemma for asynchronous session types)**.** *Let:*

$$T_0[] \parallel U_0[] \ (\xrightarrow{\tau})^n \ T_n[\sigma_n] \parallel U_n[\rho_n]$$

with $\sigma_n \neq \epsilon$, $\rho_n = \epsilon$, and $\forall i \leq n : (\sigma_i = \epsilon$ or $\rho_i = \epsilon)$. Let m be the length of σ_n. Then, there exists some $k \leq n$ such that:

1. *$\sigma_k = \epsilon$ and $\forall i > k : \sigma_i \neq \epsilon$*
2. *$\forall i \geq k : \rho_i = \epsilon$*

3. *there exist* T'_j, U'_j *(for all* $j \in k..n$*) such that:*

$$T_k[\sigma_k] \parallel U_k[\rho_k] = T'_k[] \parallel U'_k[] \;\; (\xrightarrow{\tau})^{n-k-m} \;\; T'_{n-m}[] \parallel U'_{n-m}[]$$
$$(\xrightarrow{\tau})^{m} \;\; T'_n[\sigma_n] \parallel U'_n[] \;=\; T_n[\sigma_n] \parallel U_n[]$$

where $\forall i \in 0..m : U'_{(n-m)+i} = U_n$

4. $T'_{n-m} \overset{\sigma_n}{\Longrightarrow} T_n$

Proof. (*sketch*) Let k be the greatest index such that both queues σ_k and ρ_k are empty. Since σ_n is not empty, then Items 1 and 2 follow. For Item 3, we observe that T_k performs some outputs (and no inputs, since the queues ρ_i are persistently empty for $i \geq k$). These outputs can be split in two parts: in the first one (called $\tilde{\sigma}$) the inputs are read later on by U_k, while in the second part (called $\tilde{\sigma'}$) the inputs remain enqueued (and so, $\tilde{\sigma'} = \sigma_n$). It is possible to reorder the moves without altering the length of the computation, and so that: (i) at the beginning, $\tilde{\sigma}$ is enqueued and read by U; (ii) then, the second part is enqueued, and never read. The terms T'_j, U'_j (for all $j \in k..n$) are defined according to this reordering. After the steps (i), both queues are empty, and the component U no longer moves, hence it is equal to U_n until the end of the computation.

Item 4 is a direct consequence of Item 3. □

Lemma 4.11 (Half-duplex communication in compliant async session behaviours)**.** *Let* $T \circ U$*, for some* $\circ \in \{\dashv, \vdash, \dashv\vdash, \ddot{\lhd}, \ddot{\rhd}, \ddot{\bowtie}\}$*, and assume that* $T[] \parallel U[] \Rightarrow T'[\sigma] \parallel U'[\rho]$*. Then:*

(i) $\sigma = \epsilon$ *or* $\rho = \epsilon$.

(ii) *if* $\sigma = \rho = \epsilon$ *then* $T' \circ U'$.

Proof. We first prove the statement for $\circ = \dashv$, by showing that:

$$T \dashv U \;\wedge\; T[] \parallel U[] \Rightarrow T'[\sigma] \parallel U'[\rho] \quad \text{implies} \quad \sigma = \epsilon \text{ or } \rho = \epsilon \tag{B.2}$$
$$\text{the premise above and } \sigma = \rho = \epsilon \quad \text{implies} \quad T' \dashv U' \tag{B.3}$$

We proceed by induction on the length of the sequence of τ-transitions in $T[] \parallel U[] \Rightarrow T'[\sigma] \parallel U'[\rho]$. In the base case (i.e., when there are no transitions) the thesis coincides with the hypothesis, and both Equations (B.2) and (B.3) trivially hold.

In the inductive case, let:

$$T[] \parallel U[] \;\Rightarrow\; T''[\sigma''] \parallel U''[\rho''] \;\xrightarrow{\tau}\; T'[\sigma'] \parallel U'[\rho'] \tag{B.4}$$

and by the induction hypothesis,

$$(\sigma'' = \epsilon \;\text{ or }\; \rho'' = \epsilon) \qquad \text{and} \qquad (\sigma'' = \rho'' = \epsilon \;\text{ implies }\; T'' \dashv U'') \tag{B.5}$$

We have the following cases:

- $\sigma'' \neq \epsilon$ and $\rho'' = \epsilon$. By Lemma B.9, the computation $T[] \parallel U[] \;\Rightarrow\; T''[\sigma''] \parallel U''[\rho'']$ can be rearranged as follows (while maintaining the original length):

  $$T[] \parallel U[] \;\Rightarrow\; T'''[] \parallel U''[] \;\Rightarrow\; T''[\sigma''] \parallel U''[]$$

  where $T''' \overset{\sigma''}{\Longrightarrow} T''$ (by item 4 of Lemma B.9). Since the length of the computation

  $$T[] \parallel U[] \Rightarrow T'''[] \parallel U''[]$$

  is shorter than the the length of the original computation, by the induction hypothesis we have $T''' \dashv U''$. Let $\sigma'' = \mathsf{!a}.\sigma'''$. Then, by Proposition 4.3, U'' must be (up-to unfolding) an *external* choice with a $\mathsf{?a}$-branch. Hence, the rightmost τ-transition in Equation (B.4) can only be generated in two ways:

  – via synchronisation, i.e.:

  $$T''[\mathsf{!a}.\sigma'''] \parallel U''[] \;\xrightarrow{\tau}\; T'[\sigma'''] \parallel U'[] \qquad (\text{with } \sigma''' = \sigma')$$

  where $T' = T''$. We can notice that item *(i)* of the thesis is already satisfied. For item *(ii)*, assume that $\sigma''' = \epsilon$. Then, $\mathsf{!a}$ is the only output transition from T''' to T'' — i.e., $T''' \overset{\mathsf{!a}}{\Rightarrow} T'' = T'$; and since $U'' \xrightarrow{\mathsf{?a}} U'$, from $T''' \dashv U''$, we deduce $T' \dashv U'$.

  – via buffering, i.e., for some b:

  $$T''[\mathsf{!a}.\sigma'''] \parallel U''[] \;\xrightarrow{\tau}\; T'[\mathsf{!a}.\sigma'''.\mathsf{!b}] \parallel U'[] \qquad (\text{where } U' = U'')$$

  Then, we satisfy item *(i)* and (vacuously) item *(ii)* of the thesis;

- $\sigma'' = \epsilon$ and $\rho'' \neq \epsilon$. The proof is similar to the previous case, by swapping the roles of T'' and U'', and the roles of σ and ρ.

- $\sigma'' = \rho'' = \epsilon$. Item *(i)* holds because in a single τ move, at most one of the two queues can become non-empty. Item *(ii)* holds vacuously, because the only way of reaching $\sigma' = \rho' = \epsilon$ would be through a synchronisation, which is not possible because $\sigma'' = \rho'' = \epsilon$.

This concludes the proof of Equations (B.2) and (B.3), in case $\circ = \dashv$. The other possible values of $\circ$ can be verified as follows:

- $\circ = \ddot{\lhd}$ follows by Equation (B.2) and $\dashv = \ddot{\lhd}$ (Theorem 4.9);
- $\circ \in \{\vdash, \ddot{\rhd}\}$ follow by symmetry;
- $\circ = \dashv\vdash$ holds because (by Definition 4.1) $\dashv\vdash = \vdash \cap \dashv$;
- finally, $\circ = \ddot{\bowtie}$ holds because $\ddot{\bowtie} = \ddot{\rhd} \cap \ddot{\lhd}$ (by Lemma 4.5). $\square$

Appendix C

I/O simulation

C.1 Basic properties

Lemma 6.6 ($\sqsubseteq_{\mathbb{U}_{\mathrm{ST}}}$-induced shapes of session types)**.** *$T \sqsubseteq_{\mathbb{U}_{ST}} U$ implies either (up-to unfolding):*

a. $T = U = \mathbf{0}$*;*

b. $T = \&_{k \in K} ?\mathsf{a}_k . T_k$ *and* $U = \&_{i \in I} ?\mathsf{a}_i . T_i$*, with* $\emptyset \neq I \subseteq K$ *and* $\forall i \in I \,.\, T_i \sqsubseteq_{\mathbb{U}_{ST}} U_i$*;*

c. $T = \bigoplus_{k \in K} !\mathsf{a}_k . T_k$ *and* $U = \bigoplus_{i \in I} !\mathsf{a}_i . T_i$*, with* $\emptyset \neq K \subseteq I$*, and* $\forall k \in K . T_k \sqsubseteq_{\mathbb{U}_{ST}} U_k$*.*

Proof. In the following, when no ambiguity arises, we will write $\sqsubseteq$ instead of $\sqsubseteq_{\mathbb{U}_{\mathrm{ST}}}$.

When $T \sqsubseteq U$, by Definition 6.1 we know that $\forall V \,.\, U \mathrel{\ddot{\bowtie}} V \implies T \mathrel{\ddot{\bowtie}} V$. By Theorem 4.9, this is equivalent to saying that $\forall V \,.\, U \dashv\vdash V \implies T \dashv\vdash V$. Hence, the possible forms of the pairs U, V and T, V (up-to unfolding) are given by Proposition 4.3 (case for $\dashv\vdash$). Therefore, by cases on U, we have:

$U = \mathbf{0}$. Then, we have $V = \mathbf{0}$, and $T = \mathbf{0}$.

$U = \&_{i \in I} ?\mathsf{a}_i . U_i$, with $I \neq \emptyset$. Then, we have $V = \bigoplus_{j \in J} !\mathsf{a}_j . V_j$, with $\emptyset \neq J \subseteq I$, and $\forall j \in J \,.\, U_j \dashv\vdash V_j$ (i.e., $U_j \mathrel{\ddot{\bowtie}} V_j$). Since $T \dashv\vdash V$, we also have $T = \&_{k \in K} ?\mathsf{a}_k . T_k$, with $\emptyset \neq J \subseteq K$, and $\forall j \in J \,.\, T_j \dashv\vdash V_j$ (i.e., $T_j \mathrel{\ddot{\bowtie}} V_j$). Furthermore, we have $I \subseteq K$: otherwise, $\exists i \in I \,.\, i \notin K$, and if we take a V such that $J = I$, we would have the contradiction $J \not\subseteq K$. Finally, since (as seen above) $\forall j \in J$ we have $U_j \mathrel{\ddot{\bowtie}} V_j$ and $T_j \mathrel{\ddot{\bowtie}} V_j$, we conclude $\forall j \in J \,.\, T_j \sqsubseteq U_j$;

$U = \bigoplus_{i \in I} ?\mathsf{a}_i . U_i$, with $I \neq \emptyset$. Then, we have $V = \&_{j \in J} !\mathsf{a}_j . V_j$, with $I \subseteq J$, and $\forall i \in I \,.\, U_i \dashv\vdash V_i$ (i.e., $U_i \ddot{\bowtie} V_i$). Since $T \dashv\vdash V$, we also have $T = \bigoplus_{k \in K} ?\mathsf{a}_k . T_k$, with $\emptyset \neq K \subseteq J$, and $\forall k \in K \,.\, T_k \dashv\vdash V_k$ (i.e., $T_k \ddot{\bowtie} V_k$). Furthermore, we have $K \subseteq I$: otherwise, $\exists k \in K \,.\, k \notin I$, and if we take a V such that $J = I$, we would have the contradiction $K \not\subseteq J$. Finally, since (as seen above) $\forall k \in K$ we have $U_k \ddot{\bowtie} V_k$ and $T_k \ddot{\bowtie} V_k$, we conclude $\forall k \in K \,.\, T_k \sqsubseteq U_k$.

□

Proof of Lemma 6.8 on page 49

Proof. We show that $\forall (p,q) \in (\bigcup \ddot{\mathbb{R}})$, all conditions in Definition 6.2 hold. We simply notice that when $p \,(\bigcup \ddot{\mathbb{R}})\, q$, then there exists an I/O simulation $\ddot{\mathfrak{R}} \subseteq (\bigcup \ddot{\mathbb{R}})$ such that $p \,\ddot{\mathfrak{R}}\, q$, for some predictive set $\mathbb{Q}$; using such a predictive set, clauses *a.–b.* of Definition 6.2 hold — and moreover, the pairs of reducts $(p', q') \in \ddot{\mathfrak{R}}$ from clauses *c.–e.* also belong to $(\bigcup \ddot{\mathbb{R}})$. □

C.2 On $\ddot{\leqslant}$ as a preorder for $\mathbb{U}$

Lemma C.1. *$\ddot{\leqslant}$ is reflexive.*

Proof. Consider the identity relation: $\ddot{\mathfrak{R}} = \{(p,p) \,|\, p \in \mathbb{U}\}$. We can easily verify that that $\ddot{\mathfrak{R}}$ is an I/O simulation, with $\{p\}$ being the predictive supporting each pair $p \,\ddot{\mathfrak{R}}\, p$. □

Lemma C.2 ($\ddot{\leqslant}$ and τ-moves (I))**.** *Let $p \,\ddot{\leqslant}\, q$, with predictive set $\mathbb{Q}$. Then, $p \Rightarrow p'$ implies $p' \,\ddot{\leqslant}\, q$, with predictive set $\mathbb{Q}'$ such that $\mathbb{Q} \Rrightarrow \mathbb{Q}'$.*

Proof. We proceed by induction on the length of the sequence of τ-transitions in $p \Rightarrow p'$. The base case ($n = 0$) is trivial. For the inductive case, let $p \Rightarrow p^* \xrightarrow{\tau} p'$. By the induction hypothesis, we have $p^* \,\ddot{\leqslant}\, q$ with predictive set $\mathbb{Q}^*$ such that $\mathbb{Q} \Rrightarrow \mathbb{Q}^*$. To show that $p' \,\ddot{\leqslant}\, q$, we observe that, (by item *c.* of Definition 6.2) $p^* \xrightarrow{\tau} p'$ implies that $\exists q' \,.\, \mathbb{Q}^* \Rightarrow q' \wedge p' \,\ddot{\leqslant}\, q'$; furthermore, $p' \,\ddot{\leqslant}\, q'$ is supported by some predictive set $\mathbb{Q}'$ such that $q' \Rrightarrow \mathbb{Q}'$. Let now $\ddot{\mathfrak{R}} = \ddot{\leqslant} \cup \{(p', q)\}$: we show that $\ddot{\mathfrak{R}}$ is an I/O simulation. This claim trivially holds for the subset $\ddot{\leqslant} \subseteq \ddot{\mathfrak{R}}$, and thus we only need to check whether the clauses of Definition 6.2 hold for the additional pair (p', q). Since $q \Rightarrow q'$ and $q' \Rrightarrow \mathbb{Q}'$, we have $q \Rrightarrow \mathbb{Q}'$. Then, using $\mathbb{Q}'$ as a predictive set for (p', q), we have:

item *a.*: $p'\Downarrow^{!} = \emptyset \implies \mathbb{Q}'\Downarrow^{!} = \emptyset$ (since $\mathbb{Q}'$ is a predictive set for $p' \ddot{\leqslant} q'$);

item *b.*: $\mathbb{Q}'\Downarrow^{??} \subseteq p'\Downarrow^{?}$ and $\mathbb{Q}'\Downarrow^{?} = \emptyset \implies p'\Downarrow^{?} = \emptyset$ (since $\mathbb{Q}'$ is a predictive set for $p' \ddot{\leqslant} q'$);

item *c.* let $p' \xrightarrow{\tau} p''$. Since $p' \ddot{\leqslant} q'$, we have that $\exists q''\,.\, \mathbb{Q}' \Rightarrow q'' \,\wedge\, p'' \ddot{\leqslant} q''$, which is the thesis;

item *d.* let $p' \xrightarrow{!\mathsf{a}} p''$. Since $p' \ddot{\leqslant} q'$, we have that $\exists q''\,.\, \mathbb{Q}' \overset{!\mathsf{a}}{\Rightarrow} q'' \,\wedge\, p'' \ddot{\leqslant} q''$, which is the thesis;

item *e.* let $p' \xrightarrow{?\mathsf{a}} p'' \,\wedge\, \mathbb{Q}' \xRightarrow{??\mathsf{a}}$. Since $p' \ddot{\leqslant} q'$, we have that $\exists q''\,.\, \mathbb{Q}' \overset{?\mathsf{a}}{\Rightarrow} q'' \,\wedge\, p'' \ddot{\leqslant} q''$, which is the thesis.

Therefore, $\ddot{\mathfrak{R}}$ is an I/O simulation, and $(p', q) \in \ddot{\mathfrak{R}}$.

We conclude by observing that, from $\mathbb{Q} \Rrightarrow \mathbb{Q}^*$, $\mathbb{Q}^* \Rightarrow q'$ and $q' \Rrightarrow \mathbb{Q}'$, we have $\mathbb{Q} \Rrightarrow \mathbb{Q}'$. $\square$

Lemma C.3 ($\ddot{\leqslant}$ and τ-moves (II))**.** *Let $p \ddot{\leqslant} q$, with predictive set $\mathbb{Q}$. Then, $q' \Rightarrow q$ implies $p \ddot{\leqslant} q'$, again with predictive set $\mathbb{Q}$.*

Proof. We proceed by induction on the length of the sequence of τ-transitions in $q' \Rightarrow q$. The base case $(n = 0)$ is trivial. For the inductive case, let $q' \xrightarrow{\tau} q^* \Rightarrow q$. By the induction hypothesis, we have $p \ddot{\leqslant} q^*$, with predictive set $\mathbb{Q}$. Let now $\ddot{\mathfrak{R}} = \ddot{\leqslant} \cup \{(p, q')\}$: we show that $\ddot{\mathfrak{R}}$ is an I/O simulation. This claim trivially holds for the subset $\ddot{\leqslant} \subseteq \ddot{\mathfrak{R}}$, and thus we only need to check whether the clauses of Definition 6.2 hold for the additional pair (p, q'). Since $q' \xrightarrow{\tau} q^*$ and $q^* \Rrightarrow \mathbb{Q}$, we have $q' \Rrightarrow \mathbb{Q}$. Then, using $\mathbb{Q}$ as a predictive set for (p, q'), we have:

item *a.*: $p\Downarrow^{!} = \emptyset \implies \mathbb{Q}\Downarrow^{!} = \emptyset$ (since $\mathbb{Q}$ is a predictive set for $p \ddot{\leqslant} q^*$);

item *b.*: $\mathbb{Q}\Downarrow^{??} \subseteq p\Downarrow^{?}$ and $\mathbb{Q}\Downarrow^{?} = \emptyset \implies p\Downarrow^{?} = \emptyset$ (since $\mathbb{Q}$ is a predictive set for $p \ddot{\leqslant} q^*$);

item *c.* let $p \xrightarrow{\tau} p'$. Since $p \ddot{\leqslant} q^*$, we have that $\exists q''\,.\, \mathbb{Q} \Rightarrow q'' \,\wedge\, p' \ddot{\leqslant} q''$, which is the thesis;

item *d.* let $p \xrightarrow{!\mathsf{a}} p'$. Since $p \mathrel{\ddot{\lessgtr}} q^*$, we have that $\exists q'' \,.\, \mathbb{Q} \overset{!\mathsf{a}}{\Rightarrow} q'' \wedge p' \mathrel{\ddot{\lessgtr}} q''$, which is the thesis;

item *e.* let $p \xrightarrow{?\mathsf{a}} p' \wedge \mathbb{Q}\overset{??\mathsf{a}}{\Longrightarrow}$. Since $p \mathrel{\ddot{\lessgtr}} q^*$, we have that $\exists q''\,.\, \mathbb{Q} \overset{?\mathsf{a}}{\Rightarrow} q'' \wedge p' \mathrel{\ddot{\lessgtr}} q''$, which is the thesis.

Therefore, $\ddot{\mathfrak{R}}$ is an I/O simulation; we conclude by observing that $(p, q') \in \ddot{\mathfrak{R}}$. □

Proof of Lemma 6.9 on page 49

Proof. By Lemma C.2 we have that, whenever $p \mathrel{\ddot{\lessgtr}} q$ holds, then $p \Rightarrow p'$ implies $p' \mathrel{\ddot{\lessgtr}} q$; therefore, by Lemma C.3, $q' \Rightarrow q$ implies $p' \mathrel{\ddot{\lessgtr}} q'$. □

C.3 Properties of predictive sets

Proposition C.4. *Given a set of behaviours* $\mathbb{Q}$, $?\mathsf{a} \in \mathbb{Q}\Downarrow^{??}$ *implies:*

(i) $\forall q \in \mathbb{Q} \,.\, q\overset{??\mathsf{a}}{\Longrightarrow}$

(ii) $\forall q' \,.\, \mathbb{Q} \Rightarrow q'$ *implies* $q'\overset{??\mathsf{a}}{\Longrightarrow}$.

Proof. Direct consequences of Definition 2.6. □

Proposition C.5. *Let* $p \mathrel{\ddot{\lessgtr}} q$ *with predictive set* $\mathbb{Q}$*. Then,* $\mathbb{Q}\Downarrow^{??} \subseteq p\Downarrow^{??}$.

Proof. Assume $\mathbb{Q}\overset{??\mathsf{a}}{\Longrightarrow}$, and let p' such that $p \Rightarrow p'$. By Lemma C.2, we have $p' \mathrel{\ddot{\lessgtr}} q$, with a predictive set $\mathbb{Q}'$ such that $\mathbb{Q} \Rrightarrow \mathbb{Q}'$. This, by Proposition C.4, means $\mathbb{Q}'\overset{??\mathsf{a}}{\Longrightarrow}$, and therefore (by item *b.* of Definition 6.2), $p' \overset{?\mathsf{a}}{\Rightarrow}$. Hence, by Definition 2.6, we conclude $p\overset{??\mathsf{a}}{\Longrightarrow}$. □

Proposition C.6. *Let* $p \mathrel{\ddot{\lessgtr}} q$ *with predictive set* $\mathbb{Q}$*. Then,* $q\overset{??\mathsf{a}}{\Longrightarrow}$ *implies* $\mathbb{Q}\overset{??\mathsf{a}}{\Longrightarrow}$.

Proof. By Definition 6.2, $\forall q' \in \mathbb{Q} \,.\, q \Rightarrow q'$. Therefore, by Definition 2.6, we have $q'\overset{??\mathsf{a}}{\Longrightarrow}$, and we conclude $\mathbb{Q}\overset{??\mathsf{a}}{\Longrightarrow}$. □

Corollary C.7. *Let* $p \mathrel{\ddot{\lessgtr}} q$*. Then,* $q\overset{??\mathsf{a}}{\Longrightarrow}$ *implies:*

1. $p \overset{??\mathsf{a}}{\Longrightarrow}$;

2. $p \overset{?\mathsf{a}}{\Rightarrow} p'$ *implies* $\exists q' \,.\, q \overset{?\mathsf{a}}{\Rightarrow} q'$ *and* $p' \ddot{\leqslant} q'$.

Proof. Item 1. follows from Proposition C.6 and Proposition C.5.

For item 2., let $p \Rightarrow p_0 \xrightarrow{?\mathsf{a}} p_0' \Rightarrow p'$. By Lemma C.2, $p_0 \ddot{\leqslant} q$, with a predictive set $\mathbb{Q}$ such that $\mathbb{Q} \overset{??\mathsf{a}}{\Longrightarrow}$ (by Proposition C.5); now, by item *e.* of Definition 6.2, $p_0 \xrightarrow{?\mathsf{a}} p_0'$ implies $\exists q' \,.\, \mathbb{Q} \overset{?\mathsf{a}}{\Rightarrow} q'$ and $p_0' \ddot{\leqslant} q'$. Finally, again by Lemma C.2, we conclude $p' \ddot{\leqslant} q'$. □

Lemma C.8 ("Neutral elements" in a predictive set)**.** *Let $p \ddot{\leqslant} q$, for some predictive set $\mathbb{Q}$. Then, for all q_0 such that $q \Rightarrow q_0$ and $q_0{\Downarrow}^! \subseteq \mathbb{Q}{\Downarrow}^!$, we have that $\mathbb{Q}_1 = \mathbb{Q} \cup \{q_0\}$ is still a predictive set for $p \ddot{\leqslant} q$.*

Proof. Immediate, by noticing that for all weak barbs of q_0 allowed by the hypotheses, we have:

- $\mathbb{Q}{\Downarrow}^! = \mathbb{Q}_1{\Downarrow}^!$, and therefore $p{\Downarrow}^! = \emptyset \implies \mathbb{Q}_1{\Downarrow}^! = \emptyset$, thus satisfying clause *a.* of Definition 6.2.

- $\mathbb{Q}_1{\Downarrow}^{??} \subseteq \mathbb{Q}{\Downarrow}^{??}$ — and therefore, $\mathbb{Q}_1$ does not require more inputs to p w.r.t. $\mathbb{Q}$, thus satisfying the first part of clause *b.* of Definition 6.2. For the second part, we notice that $\mathbb{Q}_1{\Downarrow}^{?} = \emptyset$ implies $\mathbb{Q}{\Downarrow}^{?} = \emptyset$, which in turn implies $p{\Downarrow}^{?} = \emptyset$;

- all states reachable from $\mathbb{Q}$ are still reachable from $\mathbb{Q}_1$ with the same transitions, thus satisfying clauses *c.*–*d.* of Definition 6.2;

- when $\mathbb{Q}{\Downarrow}^{??} \ni ?\mathsf{a} \notin \mathbb{Q}_1{\Downarrow}^{??}$, then the premise of clause *e.* of Definition 6.2 becomes false; otherwise, when $?\mathsf{a} \in \mathbb{Q}_1{\Downarrow}^{??} \cap \mathbb{Q}{\Downarrow}^{??}$, we notice (as in the previous point) that all states reachable from $\mathbb{Q}$ are still reachable from $\mathbb{Q}_1$ with the same transitions. Therefore, in both cases, clause *e.* of Definition 6.2 is satisfied.

□

Proof of Theorem 6.12 on page 50

Proof. Reflexivity follows from Lemma C.1.

In order to prove transitivity, let:

$$\ddot{\mathfrak{R}} \;=\; \big\{(p,r) \,\big|\, \exists q \,.\, p \,\ddot{\leqslant}\, q \,\wedge\, q \,\ddot{\leqslant}\, r\big\}$$

We show that $\ddot{\mathfrak{R}}$ is an I/O simulation. Let $(p,r) \in \ddot{\mathfrak{R}}$, and let q be such that $p \,\ddot{\leqslant}\, q$ and $q \,\ddot{\leqslant}\, r$. Then, let:

(i) $\ddot{\mathfrak{R}}_1$ be an I/O simulation such that $p \,\ddot{\mathfrak{R}}_1\, q$, with $\mathbb{Q}$ as predictive set. So, we have $q \Rrightarrow \mathbb{Q}$, and:

a. $p{\Downarrow}^{!} = \emptyset \implies \mathbb{Q}{\Downarrow}^{!} = \emptyset$;

b. $\mathbb{Q}{\Downarrow}^{??} \subseteq p{\Downarrow}^{?} \;\;\wedge\;\; \mathbb{Q}{\Downarrow}^{?} = \emptyset \implies p{\Downarrow}^{?} = \emptyset$;

c. $p \xrightarrow{\tau} p' \implies \exists q' \,.\, \mathbb{Q} \Rightarrow q' \;\wedge\; p' \,\ddot{\mathfrak{R}}_1\, q'$;

d. $p \xrightarrow{!\mathsf{a}} p' \implies \exists q' \,.\, \mathbb{Q} \overset{!\mathsf{a}}{\Rightarrow} q' \;\wedge\; p' \,\ddot{\mathfrak{R}}_1\, q'$;

e. $p \xrightarrow{?\mathsf{a}} p' \wedge \mathbb{Q} \overset{??\mathsf{a}}{\Longrightarrow} \implies \exists q' \,.\, \mathbb{Q} \overset{?\mathsf{a}}{\Rightarrow} q' \;\wedge\; p' \,\ddot{\mathfrak{R}}_1\, q'$;

(ii) $\ddot{\mathfrak{R}}_2$ be an I/O simulation such that $q \,\ddot{\mathfrak{R}}_2\, r$.

In the following, let I index the elements of $\mathbb{Q}$ — i.e., $\mathbb{Q} = \{q_i\}_{i \in I}$.

Before proceeding, we highlight some results and definitions that we will reuse throughout this proof.

Proposition C.9. *$\forall i \in I$, we have $q_i \,\ddot{\leqslant}\, r$.*

Proof. Since $\forall i \in I \,.\, q \Rrightarrow q_i$, the statement follows from Lemma C.2. □

Definition C.10. *$\forall i \in I$, we fix $\mathbb{R}_i$ as a predictive set supporting $q_i \,\ddot{\leqslant}\, r$. Furthermore, we define:*

$$\mathbb{R} \;=\; \bigcup_{i \in I} \mathbb{R}_i$$

Corollary C.11. $\mathbb{R}{\Downarrow}^{??} \subseteq \mathbb{Q}{\Downarrow}^{??}$.

Proof. Let $?\mathsf{a} \in \mathbb{R}{\Downarrow}^{??}$. Then, $?\mathsf{a} \in \tilde{r}{\Downarrow}^{??}$, for all $\tilde{r} \in \mathbb{R}$, and so $?\mathsf{a} \in \mathbb{R}_i{\Downarrow}^{??}$. Since $q_i \,\ddot{\leqslant}\, r$ (by Proposition C.9) with predictive set $\mathbb{R}_i$ (by Definition C.10), then by Proposition C.5 it follows that $?\mathsf{a} \in q_i{\Downarrow}^{??}$. Since the above holds for all $i \in I$, we conclude that $?\mathsf{a} \in \mathbb{Q}{\Downarrow}^{??}$. □

We now show that $\mathbb{R}$ is a predictive set for the pair $(p,r) \in \ddot{\mathfrak{R}}$, according to Definition 6.2:

item *a.*: assume that $p\Downarrow^! = \emptyset$. Then, by item *(i)a.*, we have $\mathbb{Q}\Downarrow^! = \emptyset$ — and therefore, $q_i\Downarrow^! = \emptyset$ for all $i \in I$. Since $q_i \mathrel{\ddot{\leqslant}} r$, by item *a.* of Definition 6.2, this implies that $\mathbb{R}_i\Downarrow^! = \emptyset$, for all $i \in I$. Therefore $\mathbb{R}\Downarrow^! = \emptyset$.

item *b.*: for the first part of the item, from Corollary C.11, we have $\mathbb{R}\Downarrow^{??} \subseteq \mathbb{Q}\Downarrow^{??}$. Furthermore, by item *(i)b.*, we have $\mathbb{Q}\Downarrow^{??} \subseteq p\Downarrow^?$. So, we conclude $\mathbb{R}\Downarrow^{??} \subseteq p\Downarrow^?$.

For the second part of item *b.*, assume $\mathbb{R}\Downarrow^? = \emptyset$: for all $i \in I$, this implies $\mathbb{R}_i\Downarrow^? = \emptyset$; by item *b.* of Definition 6.2, we have $\forall i \in I \,.\, q_i\Downarrow^? = \emptyset$, which in turn implies $\mathbb{Q}\Downarrow^? = \emptyset$; and from item *(i)b.*, we have $p\Downarrow^? = \emptyset$. Therefore, we conclude $p\Downarrow^? = \emptyset$.

item *c.*: we have to show that, whenever $p \xrightarrow{\tau} p'$, then $\exists r' \,.\, \mathbb{R} \Rightarrow r' \wedge (p', r') \in \ddot{\mathfrak{R}}$. From item *(i)c.* we know that whenever $p \xrightarrow{\tau} p'$, $\exists q' \,.\, \mathbb{Q} \Rightarrow q' \wedge (p', q') \in \ddot{\mathfrak{R}}_1$; therefore, for some $i \in I$, $q_i \Rightarrow q'$. Now, from Lemma C.3, we have $p' \mathrel{\ddot{\leqslant}} q_i$. Since (by Proposition C.9) we also have $q_i \mathrel{\ddot{\leqslant}} r$, we choose $r' = r$, and we conclude $(p', r') \in \ddot{\mathfrak{R}}$.

item *d.*: we have to show that, whenever $p \xrightarrow{!\mathsf{a}} p'$, then $\exists r' \,.\, \mathbb{R} \overset{!\mathsf{a}}{\Rightarrow} r'$ and $(p', r') \in \ddot{\mathfrak{R}}$. By item *(i)d.* above, we know that whenever $p \xrightarrow{!\mathsf{a}} p'$, then $\exists i \in I, q_*, q'_*, q'$ such that $\mathbb{Q} \ni q_i \Rightarrow q_* \xrightarrow{!\mathsf{a}} q'_* \Rightarrow q'$ and $p' \,\ddot{\mathfrak{R}}_1\, q'$. Since $p' \mathrel{\ddot{\leqslant}} q'$ and $q'_* \Rightarrow q'$, then by Lemma C.3 it follows that $p' \mathrel{\ddot{\leqslant}} q'_*$. By Proposition C.9 and Definition C.10, we have $q_i \mathrel{\ddot{\leqslant}} r$ with some predictive set $\mathbb{R}_i \subseteq \mathbb{R}$; and by Lemma C.2, we also have $q_* \mathrel{\ddot{\leqslant}} r$ with some predictive set $\mathbb{R}_*$ such that $\mathbb{R}_i \Rrightarrow \mathbb{R}_*$ — and therefore (by item *d.* of Definition 6.2):

$$\exists r'' \,.\, \mathbb{R}_i \Rrightarrow \mathbb{R}_* \overset{!\mathsf{a}}{\Rightarrow} r'' \wedge q'_* \mathrel{\ddot{\leqslant}} r'' \tag{C.1}$$

Combining $\mathbb{R} \Rrightarrow \mathbb{R}_i$ and Equation (C.1) above, we obtain:

$$\exists r'' \,.\, \mathbb{R} \overset{!\mathsf{a}}{\Rightarrow} r'' \wedge q'_* \mathrel{\ddot{\leqslant}} r''$$

Let $r' = r''$. Since $p' \mathrel{\ddot{\leqslant}} q'_*$ and $q'_* \mathrel{\ddot{\leqslant}} r'$, we conclude that $(p', r') \in \ddot{\mathfrak{R}}$.

item *e.*: we have to show that, whenever $p \xrightarrow{?\mathsf{a}} p'$ and $\mathbb{R}\xRightarrow{??\mathsf{a}}$, then $\exists r' \,.\, \mathbb{R} \overset{?\mathsf{a}}{\Rightarrow} r'$ and $(p', r') \in \ddot{\mathfrak{R}}$. Assume that $p \xrightarrow{?\mathsf{a}} p'$ and $\mathbb{R}\xRightarrow{??\mathsf{a}}$. By Corollary C.11, we have $\mathbb{Q}\xRightarrow{??\mathsf{a}}$. Therefore, by item *(i)e.* above, $\exists i \in I, q_*, q'_*, q'$ such that $\mathbb{Q} \ni q_i \Rightarrow q_* \xrightarrow{?\mathsf{a}} q'_* \Rightarrow q'$ and $p' \,\ddot{\mathfrak{R}}_1\, q'$. By Proposition C.9 and Definition C.10, we have $q_i \mathrel{\ddot{\leqslant}} r$ with some predictive set $\mathbb{R}_i \subseteq \mathbb{R}$; and by Lemma C.2, we also have $q_* \mathrel{\ddot{\leqslant}} r$ with some predictive set $\mathbb{R}_*$ such that $\mathbb{R}_i \Rrightarrow \mathbb{R}_*$. Therefore, by item *e.* of Definition 6.2:

$$\mathbb{R}_* \xRightarrow{??\mathsf{a}} \text{ implies } \exists r'' \,.\, \mathbb{R}_i \Rrightarrow \mathbb{R}_* \overset{?\mathsf{a}}{\Rrightarrow} r'' \wedge q'_* \ddot{\leqslant} r'' \tag{C.2}$$

Finally, we notice that since $\mathbb{R}\xRightarrow{??\mathsf{a}}$ implies $\mathbb{R}_i\xRightarrow{??\mathsf{a}}$, we have $\mathbb{R}_*\xRightarrow{??\mathsf{a}}$; this, combined with $\mathbb{R} \Rrightarrow \mathbb{R}_i$ and Equation (C.2) above, gives us:

$$\exists r'' \,.\, \mathbb{R} \overset{?\mathsf{a}}{\Rrightarrow} r'' \wedge q'_* \ddot{\leqslant} r''$$

By Lemma C.2, we have $q' \ddot{\leqslant} r''$. Let $r' = r''$. Since $p' \ddot{\leqslant} q'$ and $q' \ddot{\leqslant} r'$, we conclude that $(p', r') \in \ddot{\mathfrak{R}}$. ☐

Proposition C.12 ($\ddot{\leqslant}$ vs. $\underset{\approx}{\precsim}$)**.** $\ddot{\leqslant} \not\subseteq \underset{\approx}{\precsim} \not\subseteq \ddot{\leqslant}$.

Proof. Consider the following CCS processes, where $\mathsf{a} \neq \mathsf{b}$:

- let $P = ?\mathsf{a} + ?\mathsf{b}$ and $Q = ?\mathsf{a}$. We have $P \ddot{\leqslant} Q$ and $P \not\underset{\approx}{\precsim} Q$;
- let $P = \tau \,.\, ?\mathsf{a} \,+\, \tau \,.\, ?\mathsf{b}$ and $Q = ?\mathsf{a} + ?\mathsf{b}$. We have $P \underset{\approx}{\precsim} Q$ and $P \not\ddot{\leqslant} Q$.

☐

Proof of Theorem 6.11 on page 50

Proof. To show that $\approx \neq \ddot{\approx}$, consider Figure C.1: we have that $p_{13} \ddot{\approx} p_{14}$, but $p_{13} \not\approx p_{14}$.

To show that $\approx \subset \ddot{\approx}$, since the weak bisimulation relation is symmetric, it is enough to prove that it is an I/O simulation. Let $p \approx q$. We have that $p \xrightarrow{\ell} p'$ implies $\exists q' \,.\, q \overset{\ell}{\Rrightarrow} q' \wedge p' \approx q'$, and the converse holds for the ℓ-transitions emanating from q. From this, it is immediate to see that $p{\Downarrow}^{?} = q{\Downarrow}^{?}$ and $p{\Downarrow}^{!} = q{\Downarrow}^{!}$, and using this in an inductive argument we also obtain $p{\Downarrow}^{??} = q{\Downarrow}^{??}$. Hence, for all $(p, q) \in {\approx}$, we can satisfy clauses *a.*–*e.* in Definition 6.2 by taking as predictive set $\mathbb{Q} = \{q\}$. ☐

Lemma 6.7. $T \sqsubseteq U \iff T \sqsubseteq_{\mathbb{U}_{ST}} U$.

Proof. ($\implies$). Straightforward by Definition 6.1, since $\mathbb{U}_{\mathrm{ST}} \subseteq \mathbb{U}$.

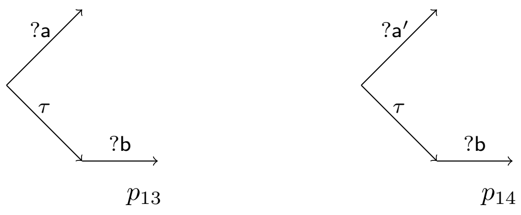


Figure C.1: Behaviours showing the difference between $\approx$ and $\ddot{\approx}$.

$(\Longleftarrow)$. Let:

$$\mathcal{R} \;=\; \{(T, r) \mid \exists U \,.\, T \sqsubseteq_{\mathbb{U}_{\mathrm{ST}}} U \,\wedge\, U \mathrel{\ddot{\bowtie}} r\} \,\cup\, \text{symmetric}$$

We show that $\mathcal{R}$ is a symmetric I/O compliance relation. For each $(T, r) \in \mathcal{R}$, there exists U such that $T \sqsubseteq_{\mathbb{U}_{\mathrm{ST}}} U$ and $U \mathrel{\ddot{\bowtie}} r$. For all such U, we have (from the proof of Lemma 4.5):

a. $U{\Downarrow}^{!} \subseteq \mathrm{co}(r{\Downarrow}^{?})$ and $r{\Downarrow}^{!} \subseteq \mathrm{co}(U{\Downarrow}^{?})$
and $\big(U{\Downarrow}^{!} = \emptyset \wedge U{\Downarrow}^{?} \neq \emptyset \implies r{\Downarrow}^{!} \neq \emptyset\big)$
and $\big(r{\Downarrow}^{!} = \emptyset \wedge r{\Downarrow}^{?} \neq \emptyset \implies U{\Downarrow}^{!} \neq \emptyset\big)$;

b. $U \xrightarrow{\ell} U' \wedge r \xrightarrow{\mathrm{co}(\ell)} r' \implies U' \mathrel{\ddot{\bowtie}} r'$;

c. $U \xrightarrow{\tau} U' \implies U' \mathrel{\ddot{\bowtie}} r$;

d. $r \xrightarrow{\tau} r' \implies U \mathrel{\ddot{\bowtie}} r'$.

Before proceeding, we also observe that, since $T \sqsubseteq_{\mathbb{U}_{\mathrm{ST}}} U$ by hypothesis, by Lemma 6.6 and $U \mathrel{\ddot{\bowtie}} r$ we have the following possibilities (up-to unfolding):

1. $U = T = \mathbf{0}$. Therefore, in this case, $r{\Downarrow} = r{\Downarrow}^{?} = r{\Downarrow}^{!} = \emptyset$;

2. $U = \&_{i \in I} ?\mathsf{a}_i . T_i$ and $T = \&_{k \in K} ?\mathsf{a}_k . T_k$, with $\emptyset \neq I \subseteq K$ and $\forall i \in I \,.\, (T_i, U_i) \in \sqsubseteq_{\mathbb{U}_{\mathrm{ST}}}$. Therefore, in this case, $\emptyset \neq r{\Downarrow}^{!} \subseteq \mathrm{co}(U{\Downarrow}^{?}) \subseteq \mathrm{co}(T{\Downarrow}^{?})$. Furthermore, $\forall i \in I . T \xrightarrow{?\mathsf{a}_i} T_i$ and $r \overset{!\mathsf{a}_i}{\Longrightarrow} r'$ implies $U_i \mathrel{\ddot{\bowtie}} r'$ (from $U \overset{?\mathsf{a}_i}{\Longrightarrow} U_i$ and Proposition B.2), while $\forall j \in K \setminus I \,.\, r \overset{!\mathsf{a}_j}{\not\Longrightarrow}$ (from $U \mathrel{\ddot{\bowtie}} r$ and Proposition B.3);

3. $U = \bigoplus_{i \in I} !\mathsf{a}_i . T_i$ and $T = \bigoplus_{k \in K} !\mathsf{a}_k . T_k$, with $\emptyset \neq K \subseteq I$, and $\forall k \in K \,.\, (T_k, U_k) \in \sqsubseteq_{\mathbb{U}_{\mathrm{ST}}}$. Therefore, in this case, $T{\Downarrow}^{!} \subseteq U{\Downarrow}^{!} \subseteq \mathrm{co}(r{\Downarrow}^{?}) \neq \emptyset$. Furthermore, $\forall k \in K \,.\, T \overset{!\mathsf{a}_k}{\Longrightarrow} T_k$ implies $r \overset{?\mathsf{a}}{\Longrightarrow}$ (from $U \overset{!\mathsf{a}_k}{\Longrightarrow} U_k$ and Proposition B.3) and $\forall r' \,.\, r \overset{?\mathsf{a}_k}{\Longrightarrow} r'$ implies $U_k \mathrel{\ddot{\bowtie}} r'$ (from $U \mathrel{\ddot{\bowtie}} r$ and Proposition B.2). Finally, by Proposition B.3, we have $r{\Downarrow}^{!} = \emptyset$.

We can now prove that $\mathcal{R}$ is a I/O compliance relation, examining the clauses of Definition 4.4:

***a.*:** we need to show that $T{\Downarrow}^{!} \subseteq \mathrm{co}(r{\Downarrow}^{?})$ and $r{\Downarrow}^{!} \subseteq \mathrm{co}(T{\Downarrow}^{?})$ and $\left(T{\Downarrow}^{!} = \emptyset \wedge T{\Downarrow}^{?} \neq \emptyset \implies r{\Downarrow}^{!} \neq \emptyset\right)$ and $\left(r{\Downarrow}^{!} = \emptyset \wedge r{\Downarrow}^{?} \neq \emptyset \implies T{\Downarrow}^{!} \neq \emptyset\right)$: these requirements are satisfied in all cases *1.*–*3.* above;

***b.*:** assume that $T \xrightarrow{\ell} T'$ and $r \xrightarrow{\mathrm{co}(\ell)} r'$. From cases *2.* and *3.* above, it follows that $\exists U' \,.\, U \overset{\ell}{\Rightarrow} U'$ and $T' \sqsubseteq_{\mathbb{U}_{\mathrm{ST}}} U' \ddot{\bowtie}\, r'$. Therefore, we conclude $(T', r') \in \mathcal{R}$ and, by symmetry of $\mathcal{R}$, $(r', T') \in \mathcal{R}$;

***c.*:** assume that $T \xrightarrow{\tau} T'$. By the semantics in Definition 2.15 we are in case *3.* above (i.e., both T and U are internal choices with multiple branches). Hence, $\exists U' \,.\, U \xrightarrow{\tau} U'$ and $T' \sqsubseteq_{\mathbb{U}_{\mathrm{ST}}} U' \ddot{\bowtie}\, r$ (by Proposition B.1). Therefore, we conclude $(T', r) \in \mathcal{R}$ and, by symmetry of $\mathcal{R}$, $(r, T') \in \mathcal{R}$;

***d.*:** assume that $r \xrightarrow{\tau} r'$. By Proposition B.1 we have $T \sqsubseteq_{\mathbb{U}_{\mathrm{ST}}} U \ddot{\bowtie}\, r'$. Therefore, we conclude $(T, r') \in \mathcal{R}$ and, by symmetry of $\mathcal{R}$, $(r', T) \in \mathcal{R}$;

Hence, $\mathcal{R}$ is a symmetric I/O compliance relation. Now, we observe that for all T, U and $r \in \mathbb{U}$, $T \sqsubseteq_{\mathbb{U}_{\mathrm{ST}}} U$ and $U \ddot{\bowtie}\, r$ imply $(T, r) \in \mathcal{R} \ni (r, T)$, and so $T \ddot{\bowtie}\, r$. By Definition 6.1, we conclude $T \sqsubseteq U$. □

C.4 On I/O simulation and I/O compliance

Proposition 6.15. *$T \ddot{\lessgtr} U \ddot{\lhd} V$ implies $T \ddot{\lhd} V$.*

Proof. We adapt the proof of Theorem 6.13 as follows:

- for clause *a.* of Definition 4.4, it suffices to proceed as in the proof of Equation (6.5) on page 52, which only relies on $\ddot{\lhd}$;
- for clause *b.*, the proof of Equation (6.5) for the sub-case $\ell = ?\mathsf{a}$ (page 52) does not hold for $\ddot{\lhd}$, because Proposition B.3 cannot be applied. We then deal with such case as follows:

  - Since $V \xrightarrow{!\mathsf{a}} V'$, by Proposition 2.18 (item *(iii)*) then V is equivalent to an internal choice. Then, by Theorem 4.9, $U \ddot{\lhd} V$ implies $U \dashv V$, and so Proposition 4.3 we have that U is a (possibly empty) external choice. By Theorem 6.16 and Lemma 6.7, $T \ddot{\leqslant} U$ implies $T \sqsubseteq_{\mathbb{U}_{\mathrm{ST}}} U$, and so by Lemma 6.6 we have that T is a larger (possibly empty) external choice. Since $T \xrightarrow{?\mathsf{a}}$, such choice is not empty, and so by Lemma 6.6 we have that also U is not empty. Since $V \xrightarrow{!\mathsf{a}}$, then by the second part of clause *a.* of Definition 4.4 it follows that $U \xrightarrow{?\mathsf{a}}$. From Definition 2.15, this implies $U \xRightarrow{??\mathsf{a}}$ — which in turn implies $\mathbb{Q} \xRightarrow{??\mathsf{a}}$. Thus, by item *e.* of Definition 6.2, $\exists U' \,.\, \mathbb{Q} \overset{?\mathsf{a}}{\Rightarrow} U' \wedge T' \ddot{\leqslant} U'$. Now, from $U \ddot{\lhd} V$ and Proposition B.2 we have $U' \ddot{\lhd} V'$: we conclude $(T', V') \in \mathcal{R}$.

- the proofs of clauses *c.* and *d.* are unmodified. □

Appendix D

Session types without types

Remark D.1. *In the following, we will only consider I/O simulations such that, for each pair of elements* $(P[\sigma], Q[\rho])$*, we have* $\sigma = \rho$*. This property is assumed for the I/O simulations given by hypothesis, and guaranteed when new I/O simulations are produced as part of a thesis.*

Notation D.2. *We write* $\{P[\sigma], Q[\sigma]\}_{\forall\sigma}$ *for* $\bigcup_{\sigma \in (\mathsf{A}^!)^*} \{P[\sigma], Q[\sigma]\}$.

Lemma D.3. *For all* σ*, let* $P[\sigma] \mathrel{\ddot{\leqslant}} Q[\sigma]$ *and* $R[\sigma] \mathrel{\ddot{\leqslant}} S[\sigma]$*, with* $(\mathrm{ins}(P) \cap \mathrm{ins}(R)) = (\mathrm{ins}(Q) \cap \mathrm{ins}(S)) = \emptyset$*. Then,* $P \mid R[\sigma] \mathrel{\ddot{\leqslant}} Q \mid S[\sigma]$*.*

Proof. From the hypothesis, let $\ddot{\mathfrak{R}}_1, \ddot{\mathfrak{R}}_2$ be I/O simulations such that for all σ, $P[\sigma] \mathrel{\ddot{\mathfrak{R}}_1} Q[\sigma]$ and $R[\sigma] \mathrel{\ddot{\mathfrak{R}}_2} S[\sigma]$. We define the following relation:

$$\ddot{\mathfrak{R}} \;=\; \Big\{ \big(P' \mid R'[\rho]\,,\, Q' \mid S'[\rho]\big) \;\Big|\; \big(P'[\rho'], Q'[\rho']\big) \in \ddot{\mathfrak{R}}_1 \;\wedge\; \big(R'[\rho''], S'[\rho'']\big) \in \ddot{\mathfrak{R}}_2 \Big\}_{\forall\rho}$$

$\ddot{\mathfrak{R}}$ is an I/O simulation, with the predictive set for each pair $(P' \mid R'[\rho]\,,\, Q' \mid S'[\rho])$ being:

$$\big\{ Q'' \mid S''[\rho] \;\big|\; Q''[\rho] \in \mathbb{Q} \;\wedge\; S''[\rho] \in \mathbb{S} \big\}$$

where $\mathbb{Q}, \mathbb{S}$ are respectively the predictive sets supporting $P'[\rho'] \mathrel{\ddot{\mathfrak{R}}_1} Q'[\rho']$ and $R'[\rho'] \mathrel{\ddot{\mathfrak{R}}_2} S'[\rho']$. The key observation is that by Proposition 6.23, P''s (resp. R''s) τ and output moves are matched by Q' (resp. S') — and the corresponding pairs of reducts, being contained in $\ddot{\mathfrak{R}}_1$ (resp. $\ddot{\mathfrak{R}}_2$), are also contained in $\ddot{\mathfrak{R}}$. The same matching also holds for input moves shared by P', Q' (resp. R', S') under clause *e.* of Definition 6.2. The only problematic situation is the following:

- $P'[\rho]\xrightarrow{?\mathsf{a}}$ and $Q'[\rho]\overset{?\mathsf{a}}{\not\Longrightarrow}$;
- $R'[\rho]\overset{?\mathsf{a}}{\Longrightarrow}$ and $S'[\rho]\overset{??\mathsf{a}}{\Longrightarrow}$;
- hence, $P' \mid R'[\rho]\xrightarrow{?\mathsf{a}}$;
- $Q' \mid S'[\rho]\overset{??\mathsf{a}}{\Longrightarrow}$.

In this case, by clause *e.* of Definition 6.2, the $?\mathsf{a}$-reduction of $P' \mid R'[\rho]$ (arising from P') would need to be I/O simulated by some weak $?\mathsf{a}$-reduction of $Q' \mid S'[\rho]$ (arising from S') — albeit the $?\mathsf{a}$-reduction of $P'[\sigma]$ was *not* I/O simulated by $Q'[\sigma]$! However, by hypothesis, we have that the inputs of P', R' and Q', S' cannot overlap; therefore, a persistent input of S' (resp. Q'), which is weakly reachable in R' (resp. P'), cannot be enabled in P' (resp. R'), and the scenario above is avoided. □

Proof of Lemma 7.5 on page 68

Proof. We recall from Remark 7.1 that all related behaviours in the statement are paired with a generic buffer $[\sigma]$ (sometimes omitted for readability), and that $P \mathrel{\ddot{\leqslant}} Q$ in $\mathbb{U}_{\mathrm{aCCS}}$ means: $\forall\sigma \,.\, P[\sigma] \mathrel{\ddot{\leqslant}} Q[\sigma]$

For rule $(+\textsc{Ctx})$, we first prove that, when $P[\sigma] \mathrel{\ddot{\leqslant}} Q[\sigma]$ (for all σ), then $\ell_\tau \,.\, P[\sigma] \mathrel{\ddot{\leqslant}} \ell_\tau \,.\, Q[\sigma]$ (for all σ). We have to consider three cases:

- $\ell_\tau = \tau$. Let $\sigma \in (\mathsf{A}^!)^*$. Then, $\ell_\tau \,.\, P[\sigma] \mathrel{\ddot{\leqslant}} \ell_\tau \,.\, Q[\sigma]$ is proved by the I/O simulation $\{(\ell_\tau \,.\, P[\sigma], \ell_\tau \,.\, Q[\sigma])\} \cup \ddot{\mathfrak{R}}$, with $\ddot{\mathfrak{R}}$ being an I/O simulation such that $P[\rho] \mathrel{\ddot{\mathfrak{R}}} Q[\rho]$ (for all ρ), and $\{\ell_\tau \,.\, Q[\sigma]\}$ being the predictive set for the additional pair;
- $\ell_\tau = ?\mathsf{a}$. The proof is similar to the previous case;
- $\ell_\tau = !\mathsf{a}$. Let $\sigma \in (\mathsf{A}^!)^*$. Then, $\ell_\tau \,.\, P[\sigma] \mathrel{\ddot{\leqslant}} \ell_\tau \,.\, Q[\sigma]$ is proved by the I/O simulation $\{(\ell_\tau \,.\, P[\sigma], \ell_\tau \,.\, Q[\sigma])\} \cup \ddot{\mathfrak{R}}$, with $\ddot{\mathfrak{R}}$ being an I/O simulation such that $P[\rho \,.\, !\mathsf{a}] \mathrel{\ddot{\mathfrak{R}}} Q[\rho \,.\, !\mathsf{a}]$ (for all ρ), and $\{\ell_\tau \,.\, Q[\sigma]\}$ being the predictive set for the additional pair.

Therefore, when for all σ we have $\forall i \in I \,.\, P_i[\sigma] \mathrel{\ddot{\leqslant}} Q_i[\sigma]$ for some I/O simulation $\ddot{\mathfrak{R}}_i$ (by the premise of rule $(+\textsc{Ctx})$), we have that $\sum_{i\in I}\ell_{\tau i} \,.\, P_i[\sigma] \mathrel{\ddot{\leqslant}} \sum_{i\in I}\ell_{\tau i} \,.\, Q_i[\sigma]$ (in the conclusion of rule (+Ctx)) holds by the following I/O simulation:

$$\left\{\left(\sum_{i\in I}\ell_{\tau i} \,.\, P_i[\sigma],\, \sum_{i\in I}\ell_{\tau i} \,.\, Q_i[\sigma]\right)\right\}_{\forall\sigma} \cup \bigcup_{i\in I}\ddot{\mathfrak{R}}_i$$

where each additional pair is supported by the predictive set $\{\sum_{i\in I}\ell_{\tau i} \,.\, Q_i[\sigma]\}$.

To prove rule $(+\tau)$, we simply observe that, for all σ, when the premise holds, $\{Q[\sigma]\}$ is a predictive set supporting the conclusion.

For rule (|L), by hypothesis we have $Q \equiv \sum_{i\in I}!\mathsf{c}_i \,.\, Q_i$, we have two I/O simulations $\ddot{\mathfrak{R}}_1, \ddot{\mathfrak{R}}_2$ such that $\forall\sigma' \,.\, P'[\sigma'] \;\ddot{\mathfrak{R}}_1\; !\mathsf{a}[\sigma']$ and $\forall\sigma' \,.\, P''[\sigma'] \;\ddot{\mathfrak{R}}_2\; ?\mathsf{b}[\sigma']$. Then, for all σ, we have that $P' \mid P''[\sigma] \mathrel{\ddot{\leqslant}} !\mathsf{a} \,.\, ?\mathsf{b} + Q[\sigma]$ is proved by the following I/O simulation:

$$\left\{\left(P'''[\sigma], !\mathsf{a} \,.\, ?\mathsf{b} + Q[\sigma]\right)\right\}_{P'''\in\left\{P_1|P_2 \;\mid\; P'\Rightarrow P_1\Rightarrow !\mathsf{a}+\cdots \,\wedge\, P''\Rightarrow P_2\Rightarrow ?\mathsf{b}+\cdots\right\}} \cup \ddot{\mathfrak{R}}_1 \cup \ddot{\mathfrak{R}}_2$$

Intuitively, P''' ranges over all the intermediate CCS$^-$ terms that $P' \mid P''[\sigma]$ may reach via τ transitions, before reaching respectively an $!\mathsf{a}$-summation and $?\mathsf{b}$-summation (which must exist — otherwise, one of the two relations in the hypotheses would be false); then, all such intermediate terms are paired with the RHS of the relation we are proving — and the predictive set for all these additional pairs is $\{?\mathsf{b}[\sigma \,.\, !\mathsf{a}]\}$.

For rule (|R), when $P[\sigma] \mathrel{\ddot{\leqslant}} Q[\sigma]$ (for all σ) and $R \equiv \sum_{i\in I}!\mathsf{c}_i \,.\, R_i$, then we have that $!\mathsf{a} \,.\, ?\mathsf{b} \,.\, P[\sigma] \mathrel{\ddot{\leqslant}} !\mathsf{a} + R \mid ?\mathsf{b} \,.\, Q[\sigma]$ is proved by the following I/O simulation:

$$\left\{\left(!\mathsf{a} \,.\, ?\mathsf{b} \,.\, P[\sigma], !\mathsf{a} + R \mid ?\mathsf{b} \,.\, Q[\sigma]\right)\right\}_{\forall\sigma} \cup \ddot{\mathfrak{R}}$$

with $\ddot{\mathfrak{R}}$ being an I/O simulation such that $?\mathsf{b} \,.\, P[\sigma \,.\, !\mathsf{a}] \;\ddot{\mathfrak{R}}\; ?\mathsf{b} \,.\, Q[\sigma \,.\, !\mathsf{a}]$ (which exists by rule (+Ctx) — see above). The predictive set for each additional pair is $\{!\mathsf{a} \mid ?\mathsf{b} \,.\, Q[\sigma]\}$.

For rule (|LR), when $P \mathrel{\ddot{\leqslant}} Q$ and $R \mathrel{\ddot{\leqslant}} S$ with $\mathrm{ins}(P) \cup \mathrm{ins}(Q) = \emptyset$, then $P \mid R[\sigma] \mathrel{\ddot{\leqslant}} Q \mid S[\sigma]$ holds by Lemma D.3.

For rule (+Int), where Q is an *output*-guarded choice, let us define:

$$\ddot{\mathfrak{R}} = \bigcup_{i\in I}\ddot{\mathfrak{R}}_i \qquad\qquad \mathbb{Q} = \bigcup_{i\in I}\mathbb{Q}_i$$

where $\ddot{\mathfrak{R}}_i$ is an I/O simulation such that $!\mathsf{a}_i \,.\, P_i[\sigma] \mathrel{\ddot{\leqslant}} !\mathsf{a}_i \,.\, Q_i[\sigma]$ (which exists by rule (+Ctx) — see above), and $\mathbb{Q}_i$ is the predictive set supporting the relation $!\mathsf{a}_i \,.\, P_i[\sigma] \; \ddot{\mathfrak{R}}_i \; !\mathsf{a}_i \,.\, Q_i[\sigma]$ (note that, by the proof of (+Ctx), we have $\mathbb{Q}_i = \{Q_i[\sigma \,.\, !\mathsf{a}_i]\}$). Then, the relation in the conclusion is proved by the following I/O simulation, where $P = \sum_{i \in I} !\mathsf{a}_i \,.\, P_i$:

$$\{(P[\sigma], Q + !\mathsf{b} \,.\, Q'[\sigma])\}_{\forall\sigma} \cup \ddot{\mathfrak{R}}$$

with $\mathbb{Q}$ being the predictive set for the additional pair (i.e., the predictive set ignores the new branch $!\mathsf{b} \,.\, Q'$).

For rule (+Ext), Q is an *input*-guarded choice. Note that whenever $j = i$, since (by premises) there exists an I/O simulation such that $P_j[\sigma] \mathrel{\ddot{\leqslant}} Q_i[\sigma]$, by rule (+Ctx) there is also an I/O simulation such that $?\mathsf{a}_\mathsf{j} \,.\, P_j[\sigma] \mathrel{\ddot{\leqslant}} ?\mathsf{a}_\mathsf{i} \,.\, Q_i[\sigma]$: therefore, let $\ddot{\mathfrak{R}}_1$ be the union of all such I/O simulations. Finally, let us consider, for each $k \in K$, an I/O simulation such that $P_k \mathrel{\ddot{\leqslant}} Q$, and let $\ddot{\mathfrak{R}}_2$ be the union of these I/O simulations. Then, the relation in the conclusion is proved by the following I/O simulation:

$$\ddot{\mathfrak{R}} \;=\; \left\{ \left(\sum_{j \in J} (?\mathsf{a}_\mathsf{j} \,.\, P_j) + \sum_{k \in K} \tau \,.\, P_k[\sigma],\; Q[\sigma] \right) \right\}_{\forall\sigma} \cup \ddot{\mathfrak{R}}_1 \cup \ddot{\mathfrak{R}}_2$$

where the predictive set for each additional pair is $\{Q[\sigma]\}$. □

Lemma 7.9. *Let P be a sequential CCS$^-$ term, with X guarded. If $P[{}^{Q}\!/_{X}] \mathrel{\ddot{\leqslant}} Q$, then $\mu_X P \mathrel{\ddot{\leqslant}} Q$. Moreover, if $\forall\sigma \,.\, P[{}^{Q}\!/_{X}][\sigma] \mathrel{\ddot{\leqslant}} Q[\sigma]$, then $\forall\sigma \,.\, \mu_X P[\sigma] \mathrel{\ddot{\leqslant}} Q[\sigma]$.*

Proof. We first prove the statement for the asynchronous semantics of CCS$^-$ (i.e., the "*moreover. . .* " part).

If $X \notin \mathrm{fv}(P)$, the thesis trivially holds. Otherwise, when $X \in \mathrm{fv}(P)$, we need to produce an I/O simulation containing the pairs $\{(\mu_X P[\sigma], Q[\sigma])\}_{\forall\sigma}$. Before proceeding, we introduce some technical machinery:

1. we write $\underline{Q}$ (i.e., Q underlined) to "tag" the occurrences of term Q which arise in $P[{}^{Q}\!/_{X}]$ due to variable substitution: e.g., if $P = ?\mathsf{a} + !\mathsf{b} \,.\, X$ and $Q = ?\mathsf{a}$, we have $P[{}^{Q}\!/_{X}] = ?\mathsf{a} + !\mathsf{b} \,.\, \underline{?\mathsf{a}}$.

2. $P\left[{}^{R}\!/\underline{Q}\right]$ is the term substitution which replaces each occurrence of $\underline{Q}$ in P with the tagged term $\underline{R}$: e.g., if $P = \mathsf{?a} + \mathsf{!b}\,.\,\underline{\mathsf{?a}}$ we have $P[{}^{R}\!/\underline{\mathsf{?a}}] = \mathsf{?a} + \mathsf{!b}\,.\,\underline{R}$ (notice that the first, untagged occurrence of $\mathsf{?a}$ is unchanged);

3. we naturally extend the semantics of async CCS so that such syntactic tags are preserved along transitions, without altering the labels. For instance, considering P and Q from item *1.* above, we have $P[{}^{Q}\!/X][] \xrightarrow{\tau} \underline{\mathsf{?a}}[\mathsf{!b}] \xrightarrow{\mathsf{?a}} \mathbf{0}[\mathsf{!b}]$ (notice that the tag is discarded when the occurrence of $\underline{Q} = \underline{\mathsf{?a}\,.\,\mathbf{0}}$ is reduced).

Now, let $\ddot{\mathcal{R}}_{\mu} = \ddot{\leqslant} \cup \ddot{\mathcal{R}}'_{\mu}$, where:

$$\ddot{\mathcal{R}}'_{\mu} \;=\; \left\{\left(P'\left[{}^{\mu_X P}\!/\underline{Q}\right][\sigma'],\, Q'[\sigma']\right) \;\middle|\; \exists\sigma_0, \sigma' \,.\, P[{}^{Q}\!/X][\sigma_0] \rightarrow^{*} P'[\sigma'] \;\wedge\; P'[\sigma'] \mathrel{\ddot{\leqslant}} Q'[\sigma']\right\}$$

Intuitively, $\ddot{\mathcal{R}}_{\mu}$ contains:

- from $\ddot{\leqslant}$: all $R[\sigma], S[\sigma] \in \mathbb{U}^{-}_{\mathrm{aCCS}}$ such that $R[\sigma] \mathrel{\ddot{\leqslant}} S[\sigma]$;
- from $\ddot{\mathcal{R}}'_{\mu}$: for all σ', all pairs $(P'[\sigma'], Q'[\sigma'])$, given by the following procedure:
  1. take $P''[\sigma']$, $Q'[\sigma']$ such that $P[{}^{Q}\!/X][\sigma] \rightarrow^{*} P''[\sigma']$ and $P''[\sigma'] \mathrel{\ddot{\leqslant}} Q'[\sigma']$ (note that since $\forall\sigma_0 \,.\, (P[{}^{Q}\!/X][\sigma_0], Q[\sigma_0])$ is in $\ddot{\leqslant}$ by hypothesis, we can find at least one of such Q' as $Q[\sigma_0] \rightarrow^{*} Q'[\sigma']$);
  2. obtain P' by replacing all occurrences of $\underline{Q}$ in P'' with $\underline{\mu_X P}$ (note that Q' is taken as-is).

We prove that $\ddot{\mathcal{R}}_{\mu}$ is an I/O simulation Let:

$$F(\mathcal{X}) = \left\{ (P''[\sigma],\, Q''[\sigma]) \;\middle|\; \exists \mathbb{Q}'' \,.\, Q''[\sigma] \Rrightarrow \mathbb{Q}'' \text{ and } \left\{\begin{array}{l}
\textbf{\textit{a.}}.\;\; P''[\sigma]{\Downarrow}^{!} = \emptyset \implies \mathbb{Q}''{\Downarrow}^{!} = \emptyset; \\[1ex]
\textbf{\textit{b.}}.\;\; \mathbb{Q}''{\Downarrow}^{??} \subseteq P''[\sigma]{\Downarrow}^{?} \;\wedge\; (\mathbb{Q}''{\Downarrow}^{?} = \emptyset \implies \\
\qquad P''[\sigma]{\Downarrow}^{?} = \emptyset); \\[1ex]
\textbf{\textit{c.}}.\;\; P''[\sigma] \xrightarrow{\tau} P'''[\sigma] \implies \exists Q'''[\sigma] \,.\, \mathbb{Q}'' \Rightarrow \\
\qquad Q'''[\sigma] \wedge (P'''[\sigma], Q'''[\sigma]) \in \mathcal{X}; \\[1ex]
\textbf{\textit{d.}}.\;\; P''[\sigma] \xrightarrow{\mathsf{!a}} P'''[\sigma'] \implies \exists Q'''[\sigma'] \,.\, \mathbb{Q}'' \overset{\mathsf{!a}}{\Rightarrow} \\
\qquad Q'''[\sigma'] \wedge (P'''[\sigma'], Q'''[\sigma']) \in \mathcal{X}; \\[1ex]
\textbf{\textit{e.}}.\;\; P''[\sigma] \xrightarrow{\mathsf{?a}} P'''[\sigma] \wedge \mathbb{Q}'' \overset{\mathsf{??a}}{\Longrightarrow} \implies \\
\qquad \exists Q'''[\sigma] \,.\, \mathbb{Q}'' \overset{\mathsf{?a}}{\Rightarrow} Q'''[\sigma] \wedge \\
\qquad (P'''[\sigma], Q'''[\sigma]) \in \mathcal{X}.
\end{array}\right. \right\}$$

By the coinduction proof principle, we have to show that $\ddot{\mathfrak{R}}_\mu \subseteq F(\ddot{\mathfrak{R}}_\mu)$. When $(P''[\sigma],\, Q''[\sigma]) \in \ddot{\mathfrak{R}}_\mu$, we have the following cases:

a. if $P''[\sigma] \mathrel{\ddot{\leqslant}} Q''[\sigma]$, the thesis is obvious: there exists a predictive set $\mathbb{Q}''$ which satisfies clauses *a.*–*e.* of F; and in particular, there exist pairs of reducts $(P'''[\sigma'], Q'''[\sigma']) \in \mathrel{\ddot{\leqslant}} \subseteq \ddot{\mathfrak{R}}_\mu$, as per clauses *c.*–*e.*. Therefore, we conclude $(P''[\sigma],\, Q''[\sigma]) \in F(\ddot{\mathfrak{R}}_\mu)$;

b. if $(P''[\sigma],\, Q''[\sigma]) \in \ddot{\mathfrak{R}}'_\mu$, then $P'' = P'\left[{}^{\mu_X P}/\underline{Q}\right]$, for some $\sigma_0, P'[\sigma]$ such that $P[{}^{Q}/X][\sigma_0] \rightarrow^* P'[\sigma] \mathrel{\ddot{\leqslant}} Q''[\sigma]$. Regarding such P', we have two possibilities:

(i) $P' \neq \underline{Q}$. By hypothesis, we have that P is sequential and X is guarded. Therefore, all occurrences of $\underline{Q}$ in P' are prefixed (since they are originated from some occurrence of X), and the same holds for all occurrences of $\underline{\mu_X P}$ in $P'\left[{}^{\mu_X P}/\underline{Q}\right] = P''$. Now, let $\mathbb{Q}''$ be a predictive set supporting $P'[\sigma] \mathrel{\ddot{\leqslant}} Q''[\sigma]$: under the prefixing considerations above, we can easily verify that $\mathbb{Q}''$ is also a predictive set for the pair $(P'\left[{}^{\mu_X P}/\underline{Q}\right][\sigma],\, Q''[\sigma]) = (P''[\sigma],\, Q''[\sigma])$. In fact, such $\mathbb{Q}''$ trivially satisfies clauses *a.* and *b.* of F, because the (weak) inputs and outputs of $P'[\sigma]$ and $P''[\sigma]$ coincide; regarding clauses *c.*–*e.*, we notice that for all $P'''[\sigma'], Q'''[\sigma']$ such that:

$$P'[\sigma] \xrightarrow{\ell_\tau} P'''[\sigma'] \quad \text{and} \quad Q''[\sigma] \Rrightarrow \mathbb{Q}'' \overset{\ell_\tau}{\Rrightarrow} Q'''[\sigma'] \quad \text{and} \quad P'''[\sigma'] \mathrel{\ddot{\leqslant}} Q'''[\sigma']$$

we have that, since $\exists \sigma_0 \,.\, P[{}^{Q}/X][\sigma_0] \rightarrow^* P'[\sigma] \xrightarrow{\ell_\tau} P'''[\sigma']$ and $P'''[\sigma'] \mathrel{\ddot{\leqslant}} Q'''[\sigma']$, $\ddot{\mathfrak{R}}'_\mu$ yields a pair $(P'''\left[{}^{\mu_X P}/\underline{Q}\right][\sigma'],\, Q'''[\sigma'])$, and therefore:

$$P''[\sigma] = P'\left[{}^{\mu_X P}/\underline{Q}\right][\sigma] \xrightarrow{\ell_\tau} P'''\left[{}^{\mu_X P}/\underline{Q}\right][\sigma'] \quad \text{and} \quad (P'''\left[{}^{\mu_X P}/\underline{Q}\right][\sigma'], Q'''[\sigma']) \in \ddot{\mathfrak{R}}'_\mu \subseteq \ddot{\mathfrak{R}}_\mu$$

Such pairs $(P'''\left[{}^{\mu_X P}/\underline{Q}\right][\sigma'], Q'''[\sigma'])$ satisfy the existentials in clauses *c.*–*e.* of F. Therefore, we conclude $(P''[\sigma],\, Q''[\sigma]) \in F(\ddot{\mathfrak{R}}_\mu)$;

(ii) $P' = \underline{Q}$. Then, we have $P'' = \underline{\mu_X P}$ and $P'[\sigma] = \underline{Q}[\sigma] \mathrel{\ddot{\leqslant}} Q''[\sigma]$. We make some observations:

1. from the pair $(P[{}^{Q}/X][\sigma],\, Q[\sigma]) \in \mathrel{\ddot{\leqslant}}$, the set $\ddot{\mathfrak{R}}'_\mu$ yields the pair:

$$\left(P[{}^{Q}/X]\left[{}^{\mu_X P}/\underline{Q}\right][\sigma],\, Q[\sigma]\right) \;=\; (P[{}^{\mu_X P}/X][\sigma],\, Q[\sigma]) \;\in\; \ddot{\mathfrak{R}}'_\mu \subseteq \ddot{\mathfrak{R}}_\mu$$

2. since recursion in $\mathbb{U}^{-}_{\text{aCCS}}$ is guarded, we have $P \neq X$ — and therefore, from case *(i)* above, $(P[{}^{\mu_X P}/X][\sigma],\, Q[\sigma]) \in F(\ddot{\mathfrak{R}}_\mu)$;

3. by applying the coinduction hypothesis, we have $(P[{}^{\mu_X P}/X][\sigma], Q[\sigma]) \in \mathrm{gfp}(F)$, i.e. $P[{}^{\mu_X P}/X][\sigma] \mathrel{\ddot{\leqslant}} Q[\sigma]$;
4. since the transition diagrams of $P''[\sigma]$ and $P[{}^{\mu_X P}/X][\sigma]$ are bisimilar, by Theorem 6.11 we have $P''[\sigma] \mathrel{\ddot{\leqslant}} P[{}^{\mu_X P}/X][\sigma]$.

Summing up:

$$P''[\sigma] \;=\; \underline{\mu_X P}[\sigma] \;\ddot{\leqslant}\; P[{}^{\mu_X P}/X][\sigma] \;\ddot{\leqslant}\; \underline{Q}[\sigma] \;\ddot{\leqslant}\; Q''[\sigma]$$

and by Theorem 6.12 (transitivity), we conclude $P''[\sigma] \mathrel{\ddot{\leqslant}} Q''[\sigma]$, thus falling back on case *a.* above — which tells us that $(P''[\sigma], Q''[\sigma]) \in F(\ddot{\mathcal{R}}_\mu)$.

Therefore, $\ddot{\mathcal{R}}_\mu$ is an I/O simulation; moreover, since $\ddot{\leqslant} \subseteq \ddot{\mathcal{R}}_\mu$ (by definition of $\ddot{\mathcal{R}}_\mu$) and $\ddot{\mathcal{R}}_\mu \subseteq \ddot{\leqslant}$ (by Definition 6.2), we have $\ddot{\mathcal{R}}_\mu = \ddot{\leqslant}$.

Finally, we are left to prove that $\forall\sigma \,.\, (\mu_X P[\sigma], Q[\sigma]) \in \ddot{\mathcal{R}}_\mu$: we simply observe that $\forall\sigma \,.\, (\mu_X P[\sigma], Q[\sigma]) \in \ddot{\mathcal{R}}'_\mu \subseteq \ddot{\mathcal{R}}_\mu$.

This concludes the proof for the asynchronous semantics of CCS$^-$. The proof for the synchronous semantics of CCS$^-$ is similar: the development above can be simplified by removing buffers and universal quantifications over σ. □

Proposition D.4 ($\ddot{\preccurlyeq}$ and large environments (I)). *For all* Γ, $X \notin \mathrm{fv}(P)$ *implies* $\big(P \mathrel{\ddot{\preccurlyeq}_\Gamma} Q \implies P \mathrel{\ddot{\preccurlyeq}_{\Gamma, X:R}} Q\big)$.

Proof. By induction on the derivation of $P \mathrel{\ddot{\preccurlyeq}_\Gamma} Q$, the result is immediate on all rules. We only mention that in the case of rule (S-μL) with recursion over X, the value of both $\Gamma(X)$ and $\Gamma'(X)$ (if defined) is "shadowed" in the new environment appearing in the premise of the rule. □

Proposition D.5 ($\ddot{\preccurlyeq}$ and large environments (II)). $P \mathrel{\ddot{\preccurlyeq}_{\Gamma, X:R}} Q \wedge X \notin \mathrm{fv}(P) \implies P \mathrel{\ddot{\preccurlyeq}_\Gamma} Q$.

Proof. By induction on the derivation of $P \mathrel{\ddot{\preccurlyeq}_\Gamma} Q$, the result is immediate on all rules. □

Definition D.6 (Term/environment substitution). *For all* P *and* Γ, *we define:*

$$P\Gamma \;=\; \begin{cases} P & \textit{if} \quad \Gamma = \emptyset \\ P[{}^{R}/X]\Gamma' & \textit{if} \quad \Gamma = \Gamma', X : R \end{cases}$$

Theorem 7.11. *Let* $P \mathrel{\ddot{\precsim}_{\Gamma}} Q$*. Then,* $\forall \sigma \,.\, P\Gamma[\sigma] \mathrel{\ddot{\lesssim}} Q[\sigma]$*.*

Proof. The statement can be proved by rule induction on Definition 7.10.

We start by examining the rules introduced in Definition 7.10:

- base case (S-**0**). We have $P[\sigma] = Q[\sigma] = \mathbf{0}[\sigma]$, and the thesis holds by Theorem 6.12 (reflexivity);
- base case (S-Var). We have $P = X$ and, on the rule premise, $\Gamma(X) = Q$. Hence, $P\Gamma[\sigma] = Q[\sigma]$, and therefore $P\Gamma[\sigma] \mathrel{\ddot{\lesssim}} Q[\sigma]$ holds by Theorem 6.12 (reflexivity);
- base case (S-X). We have $P = X$, $Q = X$ and, from the rule premise, $X \notin \operatorname{dom}(\Gamma)$. We notice that, for all σ, $P\Gamma[\sigma] = X[\sigma]$ and $Q[\sigma]$ have no transitions (by Definition 2.24): hence, we have $P\Gamma[\sigma] \sim Q[\sigma]$, and we conclude by Theorem 6.11;
- inductive case (S-μL). We have $P = \mu_X P'$ and, in the rule premise, $P' \mathrel{\ddot{\precsim}_{\Gamma'}} Q$, with $\Gamma' = \Gamma, X : Q$. Without loss of generality, by Propositions D.4 and D.5 let us assume $X \notin \operatorname{dom}(\Gamma)$. We observe:

  $$P\Gamma = (\mu_X P')\Gamma = \mu_X(P'\Gamma)$$

  By the induction hypothesis, we also have:

  $$P'\Gamma' = P'[{}^{Q}\!/\!_{X}]\Gamma = P'\Gamma[{}^{Q}\!/\!_{X}] \mathrel{\ddot{\lesssim}} Q$$

  Therefore, by Lemma 7.9, we conclude $\mu_X(P'\Gamma)[\sigma] = P\Gamma[\sigma] \mathrel{\ddot{\lesssim}} Q[\sigma]$;
- inductive case (S-μR). We have $Q = \mu_X Q'$ and, on the rule premise, $P \mathrel{\ddot{\precsim}_{\Gamma}} Q'[{}^{\mu_X Q'}\!/\!_{X}]$. By the induction hypothesis, we have $\forall \sigma \,.\, P\Gamma \mathrel{\ddot{\lesssim}} Q'[{}^{\mu_X Q'}\!/\!_{X}]$. Now, by Definition 2.24 we can verify that $Q'[{}^{\mu_X Q'}\!/\!_{X}] \sim Q$; moreover, by Definition 2.25, we can verify $\forall \sigma \,.\, Q'[{}^{\mu_X Q'}\!/\!_{X}][\sigma] \sim Q[\sigma]$. Therefore, by Theorem 6.11, we have $\forall \sigma \,.\, Q'[{}^{\mu_X Q'}\!/\!_{X}][\sigma] \mathrel{\ddot{\lesssim}} Q[\sigma]$; hence, by Theorem 6.12 (transitivity), we conclude $\forall \sigma \,.\, P\Gamma[\sigma] \mathrel{\ddot{\lesssim}} Q[\sigma]$.

We are left to examine the cases where $P \mathrel{\ddot{\precsim}_{\Gamma}} Q$ holds by some inductive rule from Table 7.2. We can observe that each one of them corresponds to an I/O simulation construction, as illustrated in the proof of Lemma 7.5: when a construction depends on the existence of some I/O simulation in the rule premises, the latter can be obtained by applying the induction hypothesis.

The only delicate cases are rules (+Int), (|R) and (|L), as discussed on page 69: while their application in derivations of non-recursive terms is always safe, we need to also examine the case that such rules are applied within recursion.

Let us now assume the critical scenario, i.e. that the rules are applied when P and Q are under recursion, and $Q[\sigma]\xRightarrow{??\mathsf{a}}$: we need to ensure $P\Gamma[\sigma]\xRightarrow{?\mathsf{a}}$. We can immediately notice that due to condition *b.* in Definition 7.7, rule (|L) cannot be applied under recursion; instead, (|R) selects a non-recursive output branch of $Q[\sigma]$ that performs no inputs: this contradicts our assumption — and therefore, (|R) cannot be applied in this scenario.

In the case of rule (+Int), suppose that we are applying (+Int) to P and Q, within a derivation relating the terms $\mu_X P'$ and Q', where Q' contains some subterm that, after unfolding via rule (S-μR), gives Q. We can verify that Q is sequential (by condition *b.* of Definition 7.7), and $?\mathsf{a}$ must precede each recursive unfolding — otherwise, we would either violate condition *a.* of Definition 7.7, or have the contradiction $Q[\sigma]\not\xRightarrow{??\mathsf{a}}$. Therefore, we have $\forall i \in I\,.\,Q_i[\sigma\,.\,!\mathsf{a}_i]\xRightarrow{??\mathsf{a}}$. By the induction hypothesis, we have $\forall i \in I\,.\,P_i\Gamma[\sigma\,.\,!\mathsf{a}_i] \mathrel{\ddot{\preccurlyeq}} Q_i[\sigma\,.\,!\mathsf{a}_i]$: this implies $\forall i \in I\,.\,?\mathsf{a} \in (P_i\Gamma[\sigma\,.\,!\mathsf{a}_i])\Downarrow^{?}$.

Due to recursion, P' is sequential (condition *b.* of Definition 7.7), and therefore $\forall \in I\,.\,P_i$ is sequential, and (by condition *a.* of Definition 7.7) either all its recursive branches contain inputs, or no branch contains inputs. Then, either:

1. for all $i \in I$, $?\mathsf{a}$ appears in P_i; or

2. for all $i \in I$, P_i only contains outputs or τs. Hence, $\forall i \in I\,.\,P_i\Gamma[\sigma\,.\,!\mathsf{a}_i]\xRightarrow{?\mathsf{a}}$ holds due to some Γ-induced substitution.

Case *1.* directly ensures $P\Gamma[\sigma]\xRightarrow{?\mathsf{a}}$, and thus the application of (+Int) is safe. Case *2.*, instead, is absurd. In fact, it can only hold within a derivation similar to the following:

$$\dfrac{P = \dfrac{\dfrac{\text{✘}\quad \Gamma(X) = Q' \neq ?\mathsf{a}\,.\,Q'}{X \mathrel{\ddot{\preccurlyeq}}_{X:Q'} ?\mathsf{a}\,.\,Q'}\;(\text{S-Var})}{!\mathsf{b}\,.\,X \mathrel{\ddot{\preccurlyeq}}_{X:Q'} !\mathsf{b}\,.\,?\mathsf{a}\,.\,Q'}\;(+\text{Int}) \qquad = Q}{\dfrac{!\mathsf{b}\,.\,X \mathrel{\ddot{\preccurlyeq}}_{X:Q'} Q'}{\mu_X !\mathsf{b}\,.\,X \mathrel{\ddot{\preccurlyeq}} \mu_Y !\mathsf{b}\,.\,?\mathsf{a}\,.\,Y}\;(\text{S-}\mu\text{L}) \qquad = Q'}\;(\text{S-}\mu\text{R})$$

which fails because, going upwards in the derivation, $\forall i \in I \,.\, P_i$ reaches the recursion variable without a syntactic occurrence of $?\mathsf{a}$ — that instead appears in each branch of Q; this contradicts the $(+\textsc{Int})$ rule premise $\forall i \in I \,.\, P_i \mathrel{\ddot{\preccurlyeq}_\Gamma} Q_i$, and thus we have the contradiction that $P \mathrel{\ddot{\preccurlyeq}_\Gamma} Q$ does *not* follow by rule $(+\textsc{Int})$. $\square$

D.1 On $\ddot{\preccurlyeq}$ as a precongruence for CCS$^-$

Lemma D.7. *Let* $\mathrm{dom}(\Gamma) \cap \mathrm{dom}(\Gamma') = \emptyset$*. Then,* $P\Gamma \mathrel{\ddot{\preccurlyeq}_{\Gamma'}} Q \iff P \mathrel{\ddot{\preccurlyeq}_{\Gamma'\Gamma}} Q$.

Proof. $\implies$ direction: by induction on Γ. In the base case $\Gamma = \emptyset$, the thesis follows immediately from the hypothesis. In the inductive case $\Gamma = \Gamma'', X : R$ (with R closed, as per Definition 7.10), by the induction hypothesis we have that $P[R/X]\Gamma'' \mathrel{\ddot{\preccurlyeq}_{\Gamma'}} Q$ implies $P[R/X] \mathrel{\ddot{\preccurlyeq}_{\Gamma',\Gamma''}} Q$, and we need to prove $P \mathrel{\ddot{\preccurlyeq}_{\Gamma',\Gamma'',X:R}} Q$. Therefore, assuming $P[R/X]\Gamma'' \mathrel{\ddot{\preccurlyeq}_{\Gamma'}} Q$, and letting $\Gamma''' = (\Gamma', \Gamma'')$, we have $P[R/X] \mathrel{\ddot{\preccurlyeq}_{\Gamma'''}} Q$ and we need to prove $P \mathrel{\ddot{\preccurlyeq}_{\Gamma''',X:R}} Q$. We proceed by induction on the rules for $P[R/X] \mathrel{\ddot{\preccurlyeq}_{\Gamma'''}} Q$:

- base case $(\textsc{s-}\mathbf{0})$. We have $P[R/X] = Q = \mathbf{0}$. This implies either:
  - $P = \mathbf{0}$. Then, by rule $(\textsc{s-}\mathbf{0})$, we conclude $P = \mathbf{0} \mathrel{\ddot{\preccurlyeq}_{\Gamma''',X:R}} \mathbf{0} = Q$;
  - $P = X$ and $R = \mathbf{0}$. Then, by rule $(\textsc{s-Var})$, we conclude $P = X \mathrel{\ddot{\preccurlyeq}_{\Gamma''',X:R}} \mathbf{0} = Q$ (since we satisfy the rule premise $(\Gamma''', X : R)(X) = R = \mathbf{0} = Q$);
- base case $(\textsc{s-}X)$. We have $P[R/X] = Q = Y$, and from the rule premise, $Y \notin \mathrm{dom}(\Gamma''')$. We have two cases:
  - $X \neq Y$. Then, $P = Y$, and by rule $(\textsc{s-Var})$ we conclude $P = Y \mathrel{\ddot{\preccurlyeq}_{\Gamma''',X:R}} Y = Q$;
  - $X = Y$, and therefore $R = X$. This case is ruled out because R cannot be an unguarded recursion variable;
- base case $(\textsc{s-Var})$. We have $P[R/X] = Y$, and from the rule premise, $\Gamma'''(Y) = Q$. This implies either:
  - $P = Y$. Then, by rule $(\textsc{s-Var})$, we conclude $P = Y \mathrel{\ddot{\preccurlyeq}_{\Gamma''',X:R}} Q$;
  - $P = X$ and $R = Y$. This case is ruled out because R cannot be an unguarded recursion variable;

- inductive case (S-μL). We have $P[{}^{R}/_{X}] = \mu_Y \cdots$. When $X = Y$, the substitution $[{}^{R}/_{X}]$ is vacuous and we conclude by Proposition D.5. Otherwise, when $Y \neq X$, we have $P[{}^{R}/_{X}] = \mu_Y(P'[{}^{R}/_{X}])$, and from the rule premise, $P'[{}^{R}/_{X}] \mathrel{\ddot{\precsim}}_{\Gamma''',Y:Q} Q$. By the rule induction hypothesis, we have $P' \mathrel{\ddot{\precsim}}_{\Gamma''',Y:Q,X:R} Q$ — and again by rule (S-μL), we conclude $\mu_Y P' \mathrel{\ddot{\precsim}}_{\Gamma''',X:R} Q$;

- inductive case (S-μR). We have $Q = \mu_Y Q'$, and from the rule premise, $P[{}^{R}/_{X}] \mathrel{\ddot{\precsim}}_{\Gamma'''} Q'[{}^{Q}/_{Y}]$. By the rule induction hypothesis, we have $P \mathrel{\ddot{\precsim}}_{\Gamma''',X:R} Q'[{}^{Q}/_{Y}]$ — and again by rule (S-μR), we conclude $P \mathrel{\ddot{\precsim}}_{\Gamma''',X:R} Q$;

- inductive cases from Table 7.2. The thesis follows by the rule induction hypothesis.

$\Longleftarrow$ direction: by induction on Γ. In the base case $\Gamma = \emptyset$, the thesis follows immediately from the hypothesis. In the inductive case $\Gamma = X : R, \Gamma''$ (where R cannot be an unguarded recursion variable, as per Definition 7.10), by the induction hypothesis we have that $P \mathrel{\ddot{\precsim}}_{\Gamma',X:R,\Gamma''} Q$ implies $P\Gamma'' \mathrel{\ddot{\precsim}}_{\Gamma',X:R} Q$, and we need to prove $P\Gamma''[{}^{R}/_{X}] \mathrel{\ddot{\precsim}}_{\Gamma'} Q$. Therefore, assuming $P \mathrel{\ddot{\precsim}}_{\Gamma',X:R,\Gamma''} Q$, we proceed by induction on the rules for $P\Gamma'' \mathrel{\ddot{\precsim}}_{\Gamma',X:R} Q$:

- base case (S-$\mathbf{0}$). We have $P\Gamma'' = Q = \mathbf{0}$. This implies either:
  - $P = \mathbf{0}$. Then, by rule (S-$\mathbf{0}$), we conclude $P\Gamma''[{}^{R}/_{X}] = \mathbf{0} \mathrel{\ddot{\precsim}}_{\Gamma''} \mathbf{0} = Q$;
  - $P = Z$ and $\Gamma''(Z) = \mathbf{0}$. Then, by rule (S-$\mathbf{0}$), we conclude $P\Gamma''[{}^{R}/_{X}] = \mathbf{0} \mathrel{\ddot{\precsim}}_{\Gamma'} \mathbf{0} = Q$;

- base case (S-X). We have $P\Gamma'' = Q = Y$, and from the rule premise, $Y \notin \operatorname{dom}(\Gamma', X : R)$. We have two cases:
  - $Y \notin \operatorname{dom}(\Gamma'')$. Then, by rule (S-$X$), we conclude $P\Gamma''[{}^{R}/_{X}] = Y \mathrel{\ddot{\precsim}}_{\Gamma'} Y = Q$;
  - $Y \in \operatorname{dom}(\Gamma'')$, and therefore $\Gamma''(Y) = Y$. This case is ruled out because Γ'' cannot associate recursion variables to unguarded recursion variables (as per Definition 7.10);

- base case (S-VAR). We have $P\Gamma'' = Y$, and from the rule premise, $(\Gamma', X : R)(Y) = Q$. This implies either:
  - $P = Y$ and $Y \notin \operatorname{dom}(\Gamma'')$. We have two cases:
    * $X \neq Y$. Then, by rule (S-VAR), we conclude $P\Gamma''[{}^{R}/_{X}] = Y \mathrel{\ddot{\precsim}}_{\Gamma'} Q$;

  - $*$ $X = Y$. Then, we have $R = Q$ — and since Q is closed, by Lemma D.9, we conclude $P\Gamma''[{}^{R}/_{X}] = Q \mathrel{\ddot{\preccurlyeq}_{\Gamma'}} Q$;

- $P = Z$, and therefore $\Gamma''(Z) = Y$. This case is ruled out because Γ'' cannot associate recursion variables to unguarded recursion variables (as per Definition 7.10);

- inductive case (S-μL). We have $P\Gamma'' = \mu_Y P'$. Without loss of generality, by Propositions D.4 and D.5 let us assume $X \neq Y \notin \mathrm{dom}\,(\Gamma') \cup \mathrm{dom}\,(\Gamma'')$. We have $\mu_Y(P'\Gamma'') \mathrel{\ddot{\preccurlyeq}_{\Gamma',X:R}} Q$, and from the rule premise, $P'\Gamma'' \mathrel{\ddot{\preccurlyeq}_{\Gamma',X:R,Y:Q}} Q$. By the rule induction hypothesis, we have $P'\Gamma''[{}^{R}/_{X}] \mathrel{\ddot{\preccurlyeq}_{\Gamma',Y:Q}} Q$; then, again by rule (S-μL), we conclude $P\Gamma''[{}^{R}/_{X}] \mathrel{\ddot{\preccurlyeq}_{\Gamma'}} Q$;
- inductive case (S-μR). We have $Q = \mu_Y Q'$, and from the rule premise, $P\Gamma'' \mathrel{\ddot{\preccurlyeq}_{\Gamma',X:R}} Q'[{}^{Q}/_{Y}]$. By the rule induction hypothesis, we have $P\Gamma''[{}^{R}/_{X}] \mathrel{\ddot{\preccurlyeq}_{\Gamma'}} Q'[{}^{Q}/_{Y}]$ — and again by rule (S-μR), we conclude $P\Gamma''[{}^{R}/_{X}] \mathrel{\ddot{\preccurlyeq}_{\Gamma'}} Q$;
- inductive cases from Table 7.2. The thesis follows by the rule induction hypothesis.

$\square$

Lemma D.8. *For all P, Γ_0, Γ such that*

(i) $\mathrm{fv}(P) \cap \mathrm{dom}\,(\Gamma_0) = \emptyset$, *and*

(ii) $\mathrm{bv}(P) \cap \mathrm{dom}\,(\Gamma) = \emptyset$, *and*

(iii) $P\Gamma$ *is a valid CCS$^-$ term,*

we have $P \mathrel{\ddot{\preccurlyeq}_{\Gamma_0,\Gamma}} P\Gamma$.

Proof. By structural induction on P:

- base case $P = \mathbf{0}$. Trivial, by rule (S-$\mathbf{0}$);
- base case $P = X$. We have two possibilities:
  - $\Gamma(X) = Q$. Then, $P\Gamma = Q$; since we also have $(\Gamma_0, \Gamma)(X) = Q$, we conclude by rule (S-Var);
  - $X \notin \mathrm{dom}\,(\Gamma)$. Then, $P\Gamma = X$; since we also have $X \notin \mathrm{dom}\,(\Gamma_0, \Gamma)$, we conclude by rule (S-X);

- inductive case $P = \sum_{i \in I} \ell_{\tau i} \,.\, P_i$. Straightforward, by applying the the induction hypothesis, and concluding via rule (+Ctx);
- inductive case $P = P' \mid P''$. Without loss of generality, by condition c. of Definition 7.7, let us assume $\mathrm{ins}(P') = \emptyset$. Since $(P' \mid P'')\Gamma = P'\Gamma \mid P''\Gamma$ is a valid CCS$^-$ term, again by condition c. of Definition 7.7, either $\mathrm{ins}(P'\Gamma) = \emptyset$ or $\mathrm{ins}(P''\Gamma) = \emptyset$: from the previous assumption, without loss of generality, let $\mathrm{ins}(P'\Gamma) = \emptyset$ (otherwise, we must have $\mathrm{ins}(P''\Gamma) = \emptyset$, and thus $\mathrm{ins}(P'') = \emptyset$, and we can perform the proof by reasoning on P'' instead of P'). By the induction hypothesis, we have:

  $$P' \mathrel{\ddot{\preccurlyeq}}_{\Gamma_0,\Gamma} P'\Gamma \quad \text{and} \quad P'' \mathrel{\ddot{\preccurlyeq}}_{\Gamma_0,\Gamma} P''\Gamma$$

  Therefore, we conclude:

  $$P = \cfrac{P' \mathrel{\ddot{\preccurlyeq}}_{\Gamma_0,\Gamma} P'\Gamma \quad P'' \mathrel{\ddot{\preccurlyeq}}_{\Gamma_0,\Gamma} P''\Gamma \quad \mathrm{ins}(P') = \mathrm{ins}(P'\Gamma) = \emptyset}{P' \mid P'' \;\mathrel{\ddot{\preccurlyeq}}_{\Gamma_0,\Gamma}\; P'\Gamma \mid P''\Gamma}\,(\mid\mathrm{LR}) = P\Gamma$$

- inductive case $P = \mu_X P'$. We have to prove:

  $$\mu_X P' \;\mathrel{\ddot{\preccurlyeq}}_{\Gamma_0,\Gamma}\; (\mu_X P')\Gamma \tag{D.1}$$

  Since $X \notin \mathrm{dom}\,(\Gamma)$, we can equivalently prove:

  $$\mu_X P' \;\mathrel{\ddot{\preccurlyeq}}_{\Gamma_0,\Gamma}\; \mu_X (P'\Gamma) \tag{D.2}$$

  Without loss of generality, by Propositions D.4 and D.5 let us assume $X \notin \mathrm{dom}\,(\Gamma_0, \Gamma)$ — and therefore, $X \notin \mathrm{dom}\,(\Gamma_0)$. By the induction hypothesis, $\forall \Gamma'_0, \Gamma'$ such that $\mathrm{fv}(P') \cap \mathrm{dom}\,(\Gamma'_0) = \emptyset$ and $\mathrm{bv}(P') \cap \mathrm{dom}\,(\Gamma') = \emptyset$, we have

  $$P' \;\mathrel{\ddot{\preccurlyeq}}_{\Gamma'_0,\Gamma'}\; P'\Gamma' \tag{D.3}$$

  If we let $\Gamma'_0 = \Gamma_0$ and $\Gamma' = (X : \mu_X P', \Gamma)$, we have:

  $$P' \;\mathrel{\ddot{\preccurlyeq}}_{\Gamma_0, X : \mu_X P', \Gamma}\; P'[{}^{\mu_X P'}/_X]\Gamma \tag{D.4}$$

Since $X \notin \mathrm{dom}(\Gamma)$, we can also write:

$$P' \;\ddot{\precsim}_{\Gamma_0,\Gamma,X:\mu_X P'}\; P'\Gamma[{}^{\mu_X P'\Gamma}/X] \tag{D.5}$$

and therefore:

$$\dfrac{\dfrac{P' \;\ddot{\precsim}_{\Gamma_0,\Gamma,X:\mu_X P'}\; P'\Gamma[{}^{\mu_X P'\Gamma}/X]}{P' \;\ddot{\precsim}_{\Gamma_0,\Gamma,X:\mu_X P'}\; \mu_X(P'\Gamma)}\,(\text{S-}\mu\text{R})}{\mu_X P' \;\ddot{\precsim}_{\Gamma_0,\Gamma}\; \mu_X(P'\Gamma)}\,(\text{S-}\mu\text{L}) \tag{D.6}$$

i.e., we proved Equation (D.2), and thus Equation (D.1) (our thesis);

□

Lemma D.9. *For all P, Γ, $(\mathrm{fv}(P) \cap \mathrm{dom}(\Gamma)) = \emptyset$ implies $P \ddot{\precsim}_\Gamma P$.*

Proof. We proceed by structural induction on P:

- base case $P = \mathbf{0}$. The thesis follows by rule (S-$\mathbf{0}$);
- base case $P = X$. Since by hypothesis we have $X \notin \mathrm{dom}(\Gamma)$, the thesis follows by rule (S-X);
- inductive case $P = \sum_{i \in I} \ell_{\tau i} \,.\, P_i$. By the induction hypothesis, we have $\forall i \in I \,.\, P_i \ddot{\precsim}_\Gamma P_i$: then, the thesis follows by rule (+CTX);
- inductive case $P = P' \mid P''$. By the induction hypothesis, we have $P' \ddot{\precsim}_\Gamma P'$ and $P'' \ddot{\precsim}_\Gamma P''$. Without loss of generality, by condition *c.* of Definition 7.7, let us assume that P' does not contain inputs: then, the thesis follows by rule (|LR);
- inductive case $P = \mu_X P'$. We have:

$$P \;=\; \dfrac{\dfrac{P' \;\ddot{\precsim}_{\Gamma,X:\mu_X P'}\; P'[{}^{\mu_X P'}/X] \quad \text{(by Lemma D.8)}}{P' \;\ddot{\precsim}_{\Gamma,X:\mu_X P'}\; \mu_X P'}\,(\text{S-}\mu\text{R})}{\mu_X P' \;\ddot{\precsim}_\Gamma\; \mu_X P'}\,(\text{S-}\mu\text{L}) \;=\; P$$

□

Definition D.10 (Environment codomain substitution)**.** *We define* $\Gamma[Q/P]$ *as the environment such that, for all* X*:*

$$\Gamma[R/Q](X) \;=\; \begin{cases} R & \text{if } \Gamma(X) \text{ is } Q \text{ or a folding/unfolding of } Q \\ \Gamma(X) & \text{otherwise} \end{cases}$$

Lemma D.11. *For all* P, Q, R, X, Γ *such that:*

(i) $\mathrm{fv}(R) \cap \mathrm{dom}\,(\Gamma) = \emptyset$, *and*

(ii) $\mathrm{bv}(P) \not\ni X \notin \mathrm{bv}(Q)$, *and*

(iii) $X \notin \mathrm{dom}\,(\Gamma)$,

$P \mathrel{\ddot{\precsim}}_{\Gamma} Q$ *implies* $P[R/X] \mathrel{\ddot{\precsim}}_{\Gamma[Q[R/X]/Q]} Q[R/X]$.

Proof. By rule induction on $P \mathrel{\ddot{\precsim}}_{\Gamma} Q$:

- base case (S-$\mathbf{0}$). We have $P = Q = \mathbf{0} = P[R/X] = Q[R/X]$, and the thesis follows again by (S-$\mathbf{0}$);
- base case (S-X). We have $P = Y = Q$, and from the rule premise, $Y \notin \mathrm{dom}\,(\Gamma)$. We need to examine two possibilities:
  - $Y = X$. Then, we need to prove $P[R/X] = R \mathrel{\ddot{\precsim}}_{\Gamma[Q[R/X]/Q]} R = Q[R/X]$. Since $\mathrm{fv}(R) \cap \mathrm{dom}\,(\Gamma) = \emptyset$ (by hypothesis *(i)*), we conclude by Lemma D.9;
  - $Y \neq X$. Then, we need to prove $P[R/X] = Y \mathrel{\ddot{\precsim}}_{\Gamma[Q[R/X]/Q]} Y = Q[R/X]$: we conclude again by rule (S-X);
- base case (S-Var). We have $P = Y \mathrel{\ddot{\precsim}}_{\Gamma} Q$, and from the rule premise, $\Gamma(Y) = Q$. We need to examine two possibilities:
  - $Y = X$. This case is absurd, because it contradicts hypothesis *(iii)*;
  - $Y \neq X$. Then, we need to prove $P[R/X] = Y \mathrel{\ddot{\precsim}}_{\Gamma[Q[R/X]/Q]} Q[R/X]$. Since $(\Gamma[Q[R/X]/Q])(Y) = Q[R/X]$, we conclude again by rule (S-Var);
- inductive cases from Table 7.2. Straightforward: for each rule, we apply the induction hypothesis on the premises, and conclude via the same rule;

- inductive case (S-μL). We have $P = \mu_Y P'$ and, from the rule premise, $P' \ddot{\precsim}_{\Gamma, Y:Q} Q$. By hypothesis *(ii)*, $P[{}^{R}/_{X}] = \mu_Y(P'[{}^{R}/_{X}])$, and thus we need to prove

$$\mu_Y(P'[{}^{R}/_{X}]) \;\ddot{\precsim}_{\Gamma\left[{}^{Q[{}^{R}/_{X}]}/_{Q}\right]}\; Q[{}^{R}/_{X}] \tag{D.7}$$

  By the induction hypothesis, we have:

$$P'[{}^{R}/_{X}] \;\ddot{\precsim}_{(\Gamma, Y:Q)\left[{}^{Q[{}^{R}/_{X}]}/_{Q}\right]}\; Q[{}^{R}/_{X}] \tag{D.8}$$

  and thus, by Definition D.10,

$$P'[{}^{R}/_{X}] \;\ddot{\precsim}_{\Gamma\left[{}^{Q[{}^{R}/_{X}]}/_{Q}\right], Y:Q[{}^{R}/_{X}]}\; Q[{}^{R}/_{X}] \tag{D.9}$$

  Therefore, again by (S-μL), we obtain Equation (D.7), which is our thesis;

- inductive case (S-μR). We have $Q = \mu_X Q'$ and, from the rule premise, $P \ddot{\precsim}_{\Gamma} Q'[{}^{\mu_Y Q'}/_{Y}]$. By hypothesis *(ii)*, $Q[{}^{R}/_{X}] = \mu_Y(Q'[{}^{R}/_{X}])$, and thus we need to prove

$$P[{}^{R}/_{X}] \;\ddot{\precsim}_{\Gamma\left[{}^{Q[{}^{R}/_{X}]}/_{Q}\right]}\; \mu_Y(Q'[{}^{R}/_{X}]) \tag{D.10}$$

  By the induction hypothesis, we have:

$$P[{}^{R}/_{X}] \;\ddot{\precsim}_{\Gamma\left[{}^{Q[{}^{R}/_{X}]}/_{Q}\right]}\; Q'[{}^{\mu_Y Q'}/_{Y}][{}^{R}/_{X}] \;=\; Q'[{}^{R}/_{X}][{}^{\mu_Y Q'[{}^{R}/_{X}]}/_{Y}] \tag{D.11}$$

  Therefore, again by (S-μR), we obtain Equation (D.10), which is our thesis.

$\square$

Lemma D.12. $\forall P, Q, X$ *s.t.* $\mathrm{bv}(P) \not\ni X \notin \mathrm{bv}(Q)$, $P \ddot{\precsim} Q$ *implies* $\mu_X P \ddot{\precsim} \mu_X Q$.

Proof. From $P \ddot{\precsim} Q$, by Lemma D.11 we have:

$$P[{}^{\mu_X Q}/_{X}] \;\ddot{\precsim}\; Q[{}^{\mu_X Q}/_{X}] \tag{D.12}$$

By Lemma D.7 ($\Longrightarrow$ direction), we obtain:

$$P \;\ddot{\precsim}_{X:\mu_X Q}\; Q[{}^{\mu_X Q}/_{X}] \tag{D.13}$$

Therefore,

$$\dfrac{\dfrac{P \;\ddot{\preccurlyeq}_{X:\mu_X Q}\; Q[{}^{\mu_X Q}/_X] \quad \text{(from D.13)}}{P \;\ddot{\preccurlyeq}_{X:\mu_X Q}\; \mu_X Q}\,\text{(S-}\mu\text{R)}}{\mu_X P \;\ddot{\preccurlyeq}\; \mu_X Q}\,\text{(S-}\mu\text{L)}$$

□

Proposition D.13. *Let $P \ddot{\preccurlyeq}_\Gamma Q$ and $R = \sum_{i \in I} \ell_{\tau i} \,.\, R_i$. Then, $R[{}^{P}/_{X}] \ddot{\preccurlyeq}_\Gamma R[{}^{Q}/_{X}]$.*

Proof. By structural induction on R. By the induction hypothesis, we have $\forall i \in I \,.\, R_i[{}^{P}/_{X}] \ddot{\preccurlyeq}_\Gamma R_i[{}^{Q}/_{X}]$ — and we conclude by rule (+Ctx). □

Lemma D.14. *Let $P \ddot{\preccurlyeq}_\Gamma Q$ and $R = R' \mid R''$. Then, if $R[{}^{P}/_{X}]$ and $R[{}^{Q}/_{X}]$ are valid CCS$^-$ terms, $R[{}^{P}/_{X}] \ddot{\preccurlyeq}_\Gamma R[{}^{Q}/_{X}]$.*

Proof. By structural induction on R. By the induction hypothesis, we have $R'[{}^{P}/_{X}] \ddot{\preccurlyeq}_\Gamma R'[{}^{Q}/_{X}]$ and $R''[{}^{P}/_{X}] \ddot{\preccurlyeq}_\Gamma R''[{}^{Q}/_{X}]$. By the restrictions on $\mathbb{U}^-_{\text{aCCS}}$ terms, we have that either R' or R'' must not contain inputs, and the context application is only valid if such a property is preserved in the resulting term after substitution. Without loss of generality, let us assume that R' does *not* contain inputs:

- if $X \notin \text{fv}(R')$, then $\text{ins}(R'[{}^{P}/_{X}]) \cup \text{ins}(R'[{}^{Q}/_{X}]) = \emptyset$;
- otherwise, if $X \in \text{fv}(R')$, we have two possibilities:
  - P, Q do *not* contain inputs. Then, we have $\text{ins}(R'[{}^{P}/_{X}]) \cup \text{ins}(R'[{}^{Q}/_{X}]) = \emptyset$;
  - otherwise, if either P or Q contains inputs, it must be the case that R'' does *not* contain free occurrences of X nor inputs — otherwise, one between $R[{}^{P}/_{X}]$ or $R[{}^{Q}/_{X}]$ would not be a valid CCS$^-$ term. Hence, we have $\text{ins}(R''[{}^{P}/_{X}]) \cup \text{ins}(R''[{}^{Q}/_{X}]) = \emptyset$.

Therefore, in all cases, we are able to satisfy the premises and side condition on inputs of rule (|LR) in Table 7.2; thus, we can conclude $R'[{}^{P}/_{X}] \mid R''[{}^{P}/_{X}] \ddot{\preccurlyeq}_\Gamma R'[{}^{Q}/_{X}] \mid R''[{}^{Q}/_{X}]$, and hence $R[{}^{P}/_{X}] \ddot{\preccurlyeq}_\Gamma R[{}^{Q}/_{X}]$. □

Conjecture D.15. *$\mu_X P \ddot{\preccurlyeq}_\Gamma Q$ implies $P[{}^{P}/_{X}] \ddot{\preccurlyeq}_\Gamma Q$.*

Proof. TBD. □

Conjecture D.16. *For all P, Q, R, Γ, Γ':*

$$P \mathrel{\ddot{\precsim}_{\Gamma}} Q \;\wedge\; Q \mathrel{\ddot{\precsim}_{\Gamma'}} R \quad\Longrightarrow\quad P \mathrel{\ddot{\precsim}_{\Gamma',\Gamma[R/Q]}} R$$

Proof. We proceed by rule induction on $P \mathrel{\ddot{\precsim}_{\Gamma}} Q$.

- base case $(\text{S-}\mathbf{0})$. We have $P = Q = \mathbf{0}$, and $Q \mathrel{\ddot{\precsim}_{\Gamma'}} R$ can only hold if $R = \mathbf{0}$. Then, again by $(\text{S-}\mathbf{0})$, we conclude $P \mathrel{\ddot{\precsim}_{\Gamma',\Gamma[R/Q]}} R$;

- base case $(\text{S-}\textsc{Var})$. We have $X = P \mathrel{\ddot{\precsim}_{\Gamma}} Q$, and from the rule premise, $\Gamma(X) = Q$. Since $(\Gamma', \Gamma[R/Q])(X) = R$, again by $(\text{S-}\textsc{Var})$, we conclude $P \mathrel{\ddot{\precsim}_{\Gamma',\Gamma[R/Q]}} R$;

- base case $(\text{S-}X)$. We have $P = X \mathrel{\ddot{\precsim}_{\Gamma}} X = Q$, and from the rule premise, $X \notin \operatorname{dom}(\Gamma)$. $Q \mathrel{\ddot{\precsim}_{\Gamma'}} R$ can hold via rules:

  - $(\text{S-}X)$. Then, $R = X$ and $X \notin \operatorname{dom}(\Gamma')$. Then, again by $(\text{S-}X)$, we conclude $P = X \mathrel{\ddot{\precsim}_{\Gamma',\Gamma[R/Q]}} X = R$;

  - $(\text{S-}\textsc{Var})$. Then, $\Gamma'(X) = R$, and from $X \notin \operatorname{dom}(\Gamma)$, we have $(\Gamma', \Gamma[R/Q])(X) = R$: by $(\text{S-}\textsc{Var})$, we conclude $P \mathrel{\ddot{\precsim}_{\Gamma',\Gamma[R/Q]}} R$;

- inductive case $(\text{S-}\mu\text{L})$. We have $P = \mu_X P' \mathrel{\ddot{\precsim}_{\Gamma}} Q$, and from the rule premise, $P' \mathrel{\ddot{\precsim}_{\Gamma, X:Q}} Q$. By the induction hypothesis, we have $P' \mathrel{\ddot{\precsim}_{\Gamma',\Gamma[R/Q],X:R}} R$, and again by rule $(\text{S-}\mu\text{L})$ we conclude $P \mathrel{\ddot{\precsim}_{\Gamma',\Gamma[R/Q]}} R$;

- inductive case $(\text{S-}\mu\text{R})$. We have $P \mathrel{\ddot{\precsim}_{\Gamma}} Q = \mu_X Q'$, and from the rule premise, $P \mathrel{\ddot{\precsim}_{\Gamma}} Q'[\mu_X Q'/X]$. Now, since $Q = \mu_X Q' \mathrel{\ddot{\precsim}_{\Gamma'}} R$, by Conjecture D.15 we also have $Q'[\mu_X Q'/X] \mathrel{\ddot{\precsim}_{\Gamma'}} R$; then, by the induction hypothesis, we have

$$P \mathrel{\ddot{\precsim}_{\Gamma',\Gamma[R/Q'[\mu_X Q'/X]]}} R$$

  Since $Q = \mu_X Q'$ is the folding of $Q'[\mu_X Q'/X]$, by Definition D.10 we have:

$$\Gamma', \Gamma\big[R/Q'[\mu_X Q'/X]\big] \;=\; \Gamma', \Gamma[R/Q]$$

  Therefore, we conclude $P \mathrel{\ddot{\precsim}_{\Gamma',\Gamma[R/Q]}} R$;

- inductive case $(|\text{L})$. We have $P' \mid P'' = P \mathrel{\ddot{\precsim}_{\Gamma}} Q = \mathsf{!a} \,.\, \mathsf{?b} + \sum_{i\in I} \mathsf{!c_i} \,.\, Q_i$, with $P' \mathrel{\ddot{\precsim}_{\Gamma}} \mathsf{!a}$ and $P'' \mathrel{\ddot{\precsim}_{\Gamma}} \mathsf{?b}$. Then, $Q \mathrel{\ddot{\precsim}_{\Gamma'}} R$ may hold because of rules:

- ($|$R), which implies $I = \emptyset$ and $R = R' \mid R''$, with $R' = \mathsf{!a} + \sum_{i \in I} \mathsf{!c_i} \,.\, R_i$ and $R'' = \mathsf{?b}$. By the restrictions of $\mathbb{U}^{-}_{\text{aCCS}}$ terms, R' cannot contain inputs, and the same is also true for P'; furthermore, we have $\mathsf{!a} \mathrel{\ddot{\preccurlyeq}_{\Gamma'}} R'$ (by rule (+Int)) and $\mathsf{?b} \mathrel{\ddot{\preccurlyeq}_{\Gamma'}} R''$ (by rule (+Ext)) — and applying the induction hypothesis, we also have $P' \mathrel{\ddot{\preccurlyeq}_{\Gamma',\Gamma[{}^{R}/_{Q}]}} R'$ and $P'' \mathrel{\ddot{\preccurlyeq}_{\Gamma',\Gamma[{}^{R}/_{Q}]}} R''$: hence, we satisfy the premises of rule ($|$LR), and we obtain $P \mathrel{\ddot{\preccurlyeq}_{\Gamma',\Gamma[{}^{R}/_{Q}]}} R$;
- (+Ctx). Then, $R = \mathsf{!a} \,.\, R'' + \sum_{i \in I} \mathsf{!c_i} \,.\, R_i$, with $R'' \mathrel{\ddot{\preccurlyeq}_{\Gamma'}} \mathsf{?b}$ and $\forall i \in I \,.\, R_i \mathrel{\ddot{\preccurlyeq}_{\Gamma'}} Q_i$. By applying the induction hypothesis, we have $P' \mathrel{\ddot{\preccurlyeq}_{\Gamma',\Gamma[{}^{R}/_{Q}]}} \mathsf{!b}$ and $P'' \mathrel{\ddot{\preccurlyeq}_{\Gamma',\Gamma[{}^{R}/_{Q}]}} \mathsf{?b}$. Thus, by rule ($|$L), we conclude $P \mathrel{\ddot{\preccurlyeq}_{\Gamma',\Gamma[{}^{R}/_{Q}]}} R$;
- (+Int): similar to case (+Ctx) above.

- inductive case ($|$R). We have $\mathsf{!a} \,.\, P' = P \mathrel{\ddot{\preccurlyeq}_{\Gamma}} Q = \mathsf{!a} + \sum_{i \in I} \mathsf{!c_i} \,.\, Q_i \mid \mathsf{?b} \,.\, Q'$, with premise $P' \mathrel{\ddot{\preccurlyeq}_{\Gamma}} \mathsf{?b} \,.\, Q'$. Then, $Q \mathrel{\ddot{\preccurlyeq}_{\Gamma'}} R$ may hold because of rules:
  - ($|$LR). Then, we have $R = R' \mid R''$, with premises $\mathsf{!a} + \sum_{i \in I} \mathsf{!c_i} \,.\, Q_i \mathrel{\ddot{\preccurlyeq}_{\Gamma'}} R'$ (which may only be due to equality, or rules (+Ctx) or (+Int)) and $\mathsf{?b} \,.\, Q' \mathrel{\ddot{\preccurlyeq}_{\Gamma'}} R''$. By applying the induction hypothesis, we have $P' \mathrel{\ddot{\preccurlyeq}_{\Gamma',\Gamma[{}^{R}/_{Q}]}} R''$, and by rule ($|$R), we conclude $P \mathrel{\ddot{\preccurlyeq}_{\Gamma',\Gamma[{}^{R}/_{Q}]}} R$;
  - ($|$L). Then, $R = \mathsf{!a}.\mathsf{?b} + \sum_{j \in J} \mathsf{!d_j} \,.\, R_j$. Furthermore, we have $\mathsf{!a} + \sum_{i \in I} \mathsf{!c_i} \,.\, Q_i \mathrel{\ddot{\preccurlyeq}_{\Gamma'}} \mathsf{!a}$ and $P' \mathrel{\ddot{\preccurlyeq}_{\Gamma}} \mathsf{?b} \,.\, Q' \mathrel{\ddot{\preccurlyeq}_{\Gamma'}} \mathsf{?b}$, which imply $\forall i \in I \,.\, \mathsf{c_i} = \mathsf{a} \;\wedge\; Q_i \mathrel{\ddot{\preccurlyeq}_{\Gamma'}} \mathbf{0}$. Hence, by applying the induction hypothesis, we have $P' \mathrel{\ddot{\preccurlyeq}_{\Gamma',\Gamma[{}^{R}/_{Q}]}} \mathsf{?b}$. By rule (+Int), we conclude $P \mathrel{\ddot{\preccurlyeq}_{\Gamma',\Gamma[{}^{R}/_{Q}]}} R$.

- inductive case (+Ctx). We have $\sum_{i \in I} \ell_{\tau i} \,.\, P_i = P \mathrel{\ddot{\preccurlyeq}_{\Gamma}} Q = \sum_{i \in I} \ell_{\tau i} \,.\, Q_i$, with $P_i \mathrel{\ddot{\preccurlyeq}_{\Gamma}} Q_i$ for all $i \in I$. We have two cases:
  - when $Q \mathrel{\ddot{\preccurlyeq}_{\Gamma'}} R$ is due to (+Ctx) or (+Int) or (+Ext), by applying the induction hypothesis we obtain $P \mathrel{\ddot{\preccurlyeq}_{\Gamma',\Gamma[{}^{R}/_{Q}]}} R$ by the same rule;
  - when the top-level choice of P and Q has a single branch with $\ell_{\tau} = \mathsf{!a}$, we have $P = \mathsf{!a} \,.\, P'$ and $Q = \mathsf{!a} \,.\, Q'$, with $P' \mathrel{\ddot{\preccurlyeq}_{\Gamma}} Q'$. In this case, $Q \mathrel{\ddot{\preccurlyeq}_{\Gamma'}} R$ may also be due to rule ($|$R). Then, we have $R = \mathsf{!a} + \sum_{j \in j} \mathsf{!c_j} \,.\, R_j \mid R'$, with $Q' \mathrel{\ddot{\preccurlyeq}_{\Gamma'}} R' = \mathsf{?b} \,.\, R''$. By applying the induction hypothesis, we have $P' \mathrel{\ddot{\preccurlyeq}_{\Gamma',\Gamma[{}^{R}/_{Q}]}} R'$, and then $P \mathrel{\ddot{\preccurlyeq}_{\Gamma',\Gamma[{}^{R}/_{Q}]}} R$, again by rule ($|$R).

- inductive case (+Int). We have two cases:
  - when $Q \mathrel{\ddot{\preccurlyeq}_{\Gamma'}} R$ is due to (+Ctx) or (+Int), by applying the induction hypothesis we obtain $P \mathrel{\ddot{\preccurlyeq}_{\Gamma',\Gamma[{}^{R}/_{Q}]}} R$ by the same rule;

- when the top-level choice of P and Q has a single branch, $Q \ddot{\preccurlyeq}_{\Gamma'} R$ may also be due to rule (|R). This is handled similarly to case (+Ctx) above, when $Q \ddot{\preccurlyeq}_{\Gamma'} R$ holds by rule (|R): also in this case, by applying the induction hypothesis, we have $P \ddot{\preccurlyeq}_{\Gamma',\Gamma[R/Q]} R$ by rule (|R).

- inductive case (+Ext). $Q \ddot{\preccurlyeq}_{\Gamma'} R$ may be due to (+Ctx) or (+Ext); by the induction hypothesis, we obtain $P \ddot{\preccurlyeq}_{\Gamma',\Gamma[R/Q]} R$ from the same rule.

- inductive case (|LR). We have $P = P' \mid P'' \ddot{\preccurlyeq}_{\Gamma} Q' \mid Q'' = Q$, with $P' \ddot{\preccurlyeq}_{\Gamma} Q'$, $P'' \ddot{\preccurlyeq}_{\Gamma} Q''$, and $\mathrm{ins}(P') \cup \mathrm{ins}(Q') = \emptyset$. When $Q \ddot{\preccurlyeq}_{\Gamma'} R$ holds, we have the following cases:

  - if it holds by rule (|LR), we have $R = R' \mid R''$ with $Q' \ddot{\preccurlyeq}_{\Gamma'} R'$, $Q'' \ddot{\preccurlyeq}_{\Gamma'} R''$, and $\mathrm{ins}(Q') \cup \mathrm{ins}(R') = \emptyset$. By applying the induction hypothesis, we have $P \ddot{\preccurlyeq}_{\Gamma',\Gamma[R/Q]} R$ (again by rule (|LR));
  - if it holds by rule (|L), we have $Q' \ddot{\preccurlyeq}_{\Gamma'} \mathsf{!a}$, $Q'' \ddot{\preccurlyeq}_{\Gamma'} \mathsf{?b}$ and $R = \mathsf{!a} \,.\, \mathsf{?b} + R''$. Hence, from $P' \ddot{\preccurlyeq}_{\Gamma} Q'$ and $P'' \ddot{\preccurlyeq}_{\Gamma} Q''$, by applying the induction hypothesis we have $P' \ddot{\preccurlyeq}_{\Gamma',\Gamma[R/Q]} \mathsf{!a}$ and $P'' \ddot{\preccurlyeq}_{\Gamma',\Gamma[R/Q]} \mathsf{?b}$. Therefore, we can use these premises to apply rule (|L) and obtain $P \ddot{\preccurlyeq}_{\Gamma',\Gamma[R/Q]} R$.

- inductive case (+τ). We have $P = \sum_{i \in I} \tau \,.\, P_i \ddot{\preccurlyeq}_{\Gamma} Q$, with $\forall i \in I \,.\, P_i \ddot{\preccurlyeq}_{\Gamma} Q$. When $Q \ddot{\preccurlyeq}_{\Gamma'} R$, By applying the induction hypothesis we have $\forall i \in I \,.\, P_i \ddot{\preccurlyeq}_{\Gamma',\Gamma[R/Q]} R$. Then, $P \ddot{\preccurlyeq}_{\Gamma',\Gamma[R/Q]} R$, again by rule (+$\tau$).

□

Conjecture D.17. *$\ddot{\preccurlyeq}$ is a preorder for $\mathbb{U}^{-}_{aCCS}$ terms.*

Proof. Reflexivity holds by Lemma D.9; transitivity holds by Conjecture D.16. □

Conjecture 10.2. *$\ddot{\preccurlyeq}$ is a precongruence for CCS$^-$.*

Proof. From Conjecture D.17, we have that $\ddot{\preccurlyeq}$ is a preorder for $\mathbb{U}^{-}_{\mathrm{aCCS}}$ terms; from Lemma D.12, Proposition D.13 and Lemma D.14 we have that $\ddot{\preccurlyeq}$ is preserved by all operators of $\mathbb{U}^{-}_{\mathrm{aCCS}}$. We conclude that $\ddot{\preccurlyeq}$ is a precongruence for $\mathbb{U}^{-}_{\mathrm{aCCS}}$ terms. □